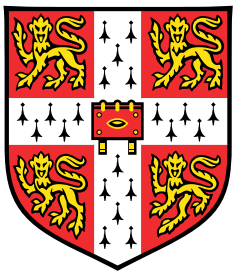
Converting to Optimization in Machine Learning: Perturb-and-MAP, Differential Privacy, and Program Synthesis



Matej Balog

Department of Engineering
University of Cambridge

This thesis is submitted for the degree of
Doctor of Philosophy

Corpus Christi College February 2020

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Matej Balog
February 2020

Converting to Optimization in Machine Learning: Perturb-and-MAP, Differential Privacy, and Program Synthesis

Matej Balog

On a mathematical level, most computational problems encountered in machine learning are instances of one of four abstract, fundamental problems: *sampling*, *integration*, *optimization*, and *search*. Thanks to the rich history of the respective mathematical fields, disparate methods with different properties have been developed for these four problem classes. As a result it can be beneficial to convert a problem from one abstract class into a problem of a different class, because the latter might come with insights, techniques, and algorithms well suited to the particular problem at hand. In particular, this thesis contributes four new methods and generalizations of existing methods for converting specific non-optimization machine learning tasks into optimization problems with more appealing properties.

The first example is partition function estimation (an *integration* problem), where an existing algorithm – the *Gumbel trick* – for converting to the MAP *optimization* problem is generalized into a more general family of algorithms, such that other instances of this family have better statistical properties. Second, this family of algorithms is further generalized to another *integration* problem, the problem of estimating Rényi entropies. The third example shows how an intractable *sampling* problem arising when wishing to publicly release a database containing sensitive data in a safe ("differentially private") manner can be converted into an *optimization* problem using the theory of Reproducing Kernel Hilbert Spaces. Finally, the fourth case study casts the challenging discrete *search* problem of program synthesis from input-output examples as a supervised learning task that can be efficiently tackled using gradient-based *optimization*.

In all four instances, the conversions result in novel algorithms with desirable properties. In the first instance, new generalizations of the Gumbel trick can be used to construct statistical estimators of the partition function that achieve the same estimation error while using up to 40% fewer samples. The second instance shows that unbiased estimators of the Rényi entropy can be constructed in the Perturb-and-MAP framework. The main contribution of the third instance is theoretical: the conversion shows that it is possible to construct an algorithm for releasing synthetic databases that approximate databases containing sensitive data in a mathematically precise sense,

and to prove results about their approximation errors. Finally, the fourth conversion yields an algorithm for synthesising program source code from input-output examples that is able to solve test problems 1-3 orders of magnitude faster than a wide range of baselines.

Acknowledgements

My biggest thanks go to my family and friends who have supported me on this journey. Its academic component also would not have been possible without the dedicated teachers at Gymnázium Grösslingová in Bratislava, and the brilliant tutors and lecturers at Merton College and the Mathematics, Statistics, and Computer Science departments in Oxford.

I would like to thank Zoubin Ghahramani, Bernhard Schölkopf, and Carl E. Rasmussen for giving me the opportunity to embark on this PhD programme, and for supporting me along the way, providing both guidance and freedom as needed. I'm particularly grateful to Bernhard Schölkopf for the unique chance to visit the world-class Max-Planck-Institute in Intelligent Systems in Tübingen, and for making sure I had a valuable research experience there.

A most enriching part of the PhD programme have been collaborations with many wonderful colleagues – Nilesh Tripuraneni, Rishabh Singh, and Danny Tarlow – among many others. Thank you all!

Finally, I would like to highlight friends who have supported me all the way along this journey both personally and academically: Tuan Anh Le, George Hron, thank you.

Ďakujem, köszönom, cảm ơn, děkuji, dziękuję, mulțumesc, danke, gracias!

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Machine learning has received an increasing amount of attention in recent years, partly thanks to successful applications of *deep learning* (a subfield of machine learning, LeCun et al., 2015) in disparate areas such as speech recognition (Hinton et al., 2012), computer vision (Krizhevsky et al., 2012), natural language understanding (Mikolov et al., 2013) and translation (Bahdanau et al., 2015), and many others. Breakthroughs in building autonomous agents for games such as Atari (Mnih et al., 2015), Go (Silver et al., 2016) and Starcraft (Vinyals et al., 2019) that match or even far surpass human performance have fuelled renewed interest in the aspiration of building "human-imitative" (Jordan, 2019) artificial intelligence (AI), to the extent that the terms AI and machine learning are nowadays sometimes conflated.

While some of the modern advances are due to the availability of larger *datasets* and to the use of modern accelerator *hardware* such as GPUs (LeCun et al., 2015), the aforementioned successes would not have been possible without having access to *methodology* (principles, techniques, and algorithms) that had been developed many years prior in a wide variety of fields including physics, mathematics, statistics, and computer science, and which continues to be developed to the present day.

This thesis is an attempt at usefully contributing to machine learning methodology, in the form of proposing new algorithms (new methods and generalizations of existing methods) for solving four important tasks. The thesis recognizes that most subtasks encountered within machine learning fall into one of four categories – *integration*, *sampling*, *optimization*, and *search* – and the common aspect of the proposed algorithms is that they convert specific *non*-optimization problems into optimization problems that are in some sense easier to solve than the original problem. The four tasks addressed in this thesis are (1) the *integration* problem of estimating normalizing constants

of distributions, (2) the *integration* problem of estimating Rényi entropies, (3) the *sampling* problem arising when wishing to release a database with sensitive data in a differentially private manner, and (4) the *search* problem of automatic program source code synthesis from an input-output example specification.

1.1 Non-technical Introduction

The purpose of this section is to explain the context and purpose of this thesis to a non-technical audience. It can be safely skipped by experts.

Machine learning Machine learning is a scientific field on the boundary of computer science and statistics. It is concerned with computational methods that attempt to extract useful knowledge from data. A cartoon example of how this is different from classical computer science is provided by the following task:

> *Build an algorithm that can be shown an image containing an apple or a banana, and the algorithm responds with "apple" or "banana" depending on which of the two is actually in the picture.*

A *non*-machine-learning approach would come up with *rules* that the computer can use to distinguish apples from bananas, based on our human understanding of the differences between the two. For example, one might try using colour as such a distinguishing rule, classifying the image as containing a banana if it contains a group of yellow pixels next to each other. This can fail for a multitude of reasons, including the fact that apples can also be yellow, or the yellow group of pixels could simply be a table in the background. One might try to persevere and come up with better distinguishing rules such as trying to recognize whether there is a round (apple) or an elongated (banana) object in the picture, but again without a lot of care this can fail due to camera angles or other objects occluding part of the fruit's shape. The point is that to build a reliable system in this way, one would have to:

1. *think hard* about the problem, and come up with more and more problem-specific rules to cover different corner cases;

2. *write computer code* that implements the hand-designed rules. The more specific a rule is to the problem at hand, the higher the chances that new complex computer code needs to be written (as opposed to reusing an already existing library).

As a result, this approach comes with a high human cost – both in terms of mental effort, and time spent developing new computer code.

The machine learning approach does away with much of this human cost, but the price one needs to pay is *data*. In our example, one needs to collect a dataset of *labelled* images of apples and bananas, i.e. a dataset where for each image it is known whether it contains an apple, or a banana. (This does not solve our task yet, because we still need to build an algorithm able to also classify images that are not part of this dataset.) *Supervised* machine learning (currently the most widely applied subtype of machine learning) essentially provides a generic recipe for automatically learning a classification rule that works well on the labelled dataset. If set up well, this rule can then be also applied to images for which the answer is not yet known, in order to *predict* it.

Probability distributions Since machine learning algorithms are built to make predictions about unknown quantities, the language of probability plays an important role. A basic concept in probability is that of a probability *distribution*. As a common example from the everyday world, a fair die defines a probability distribution that places probability $\frac{1}{6}$ on each of the numbers 1, 2, 3, 4, 5, 6. However, the distributions encountered in machine learning can be much more complicated than that – think of an unfair die that is more likely to land on some side than others, and which instead of 6 sides has millions, or even an infinite number of sides it can land on.

Fundamental problems Most machine learning algorithms can be viewed as consisting of one or more building blocks, where each block is solving one of the following four abstract, fundamental problems:

1. *Integration*: summing a large, or infinite list of numbers.
   For example, estimating the sum of numbers written on each side of the die.

2. *Sampling*: obtaining a random sample from a given probability distribution.
   For example, simulating a throw of the (possibly unfair) die.

3. *Optimization*: finding the maximum (highest point) of some function.
   For example, finding the side of the die with the largest number.

4. *Search*: finding an element in a large set that satisfies some requirement.
   For example, finding a solution to a Sudoku puzzle.

Each of these tasks can be a very difficult problem in itself, generally because there are too many elements to sum up (in integration), to choose from (in sampling), or to consider (in optimization and search).

Conversions Remarkably, there are important cases where it is possible to "convert" a problem of one of the four fundamental types into a problem of another type. For example, when wishing to sample from a probability distribution it is possible to convert the problem in such a way that instead it suffices to solve an optimization problem. Such ability to convert between different types of problems can be useful, for example because

- after conversion, the resulting new problem may have a special structure that can be exploited by specialized algorithms for the new problem;
- by performing a conversion, one can unlock new trade-offs: we will see an example where converting a sampling problem into an optimization problem allows trading off computation time for accuracy, which means that if one cannot afford to wait for the final result, it is possible to stop the algorithm early and extract a slightly less accurate but still useful answer.

This thesis contributes four new ways (or generalizations of previously known ways) to convert a specific non-optimization problem into an optimization problem.

1.2 Fundamental Problems

1.2.1 Optimization

Optimization refers to the general problem of finding an *extremum* (*minimum* or *maximum*) of a scalar-valued function $f : \mathcal{X} \to \mathbb{R} \cup \{-\infty, \infty\}$ on some domain $\mathcal{X}$. The following notation is common, stated here for maximization:

$$f^* = \max_{x \in \mathcal{X}} f(x), \qquad x^* = \operatorname*{argmax}_{x \in \mathcal{X}} f(x). \tag{1.1}$$

A primary differentiator of optimization problems, and the mathematical field of optimization as a whole, is whether the domain $\mathcal{X}$ is discrete or continuous.

Discrete optimization On finite domains an optimization problem can be solved trivially without any assumptions about f in $\mathcal{O}(|\mathcal{X}|)$ steps by considering all domain elements one by one. When $|\mathcal{X}|$ is prohibitively large or (countably) infinite, assumptions about f must be made, usually by relating its behaviour to an assumed structure of $\mathcal{X}$.

A type of structure particularly relevant for this thesis is a *discrete graphical model*, representing the possible values of n random variables in an undirected graphical model (see also Section 1.5.1):

- the domain factorizes as $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, where each $\mathcal{X}_i$ represents the (discrete) domain of *variable* i in the graphical model,
- a function f of interest is a potential function of the form $f(\mathbf{x}) = \sum_{F \in \mathcal{F}} \phi_F(x_F)$, where $\mathcal{F}$ is a collection of subsets of the variables $\{1, \ldots, n\}$, also called *factors*. (The factors correspond to maximal cliques in the graphical model.)

The potential function f of a graphical model defines its associated *Gibbs* probability distribution $p(\mathbf{x}) \propto \exp(f(\mathbf{x}))$, and it's easy to see that the problem of maximizing the potential function f is equivalent to finding the maximum-probability configuration under the Gibbs distribution. The latter problem is sometimes referred to as the *MAP* ("maximum a posteriori") problem, and it has been studied heavily in the literature (for example, Boykov et al., 2001; Kolmogorov, 2006; Sontag et al., 2008). The algorithms presented in Chapters 2 and 3 turn integration problems into this MAP problem and then exploit existing MAP solvers.

Other types of assumptions on the function being optimized can be made. One important notion is *submodularity*, which ordinarily encodes the intuitive notion that a non-negative set function has the "diminishing returns" property as more items are added into the set. In the context of discrete graphical models, a potential function ϕ_F on $\mathcal{X}_F := \prod_{i \in F} \mathcal{X}_i$ is submodular if

$$\forall y_F, z_F \in \mathcal{X}_F \qquad \phi_F(y_F \wedge z_F) + \phi_F(y_F \vee z_F) \leq \phi_F(y_F) + \phi_F(z_F), \tag{1.2}$$

where $\wedge$ and $\vee$ respectively denote elementwise minima and elementwise maxima across variables in the factor F. Making such submodularity assumptions can lead to faster algorithms including to the MAP problem (Darbon, 2009). The algorithms presented in Chapters 2 and 3 have the property that they preserve submodularity properties of the discrete graphical model (if the original discrete graphical model is submodular, then so is the MAP problem generated by our conversion).

Continuous optimization Although "continuous" here refers to the domain $\mathcal{X}$ being continuous, one also at the very least assumes continuity of f, lest the optimization problem become hopeless. In most applications the function f is in fact assumed differentiable (at least once, almost everywhere), and its gradient ∇f is used to inform the optimization process (a *first-order*, or *gradient-based* optimization method). When the function is twice differentiable and computing with its *Hessian* (matrix of second derivatives) is computationally tractable, such *second-order* methods can be very effective (Dennis Jr and Schnabel, 1996).

A main differentiator within the field of continuous optimization however, is whether the function f is *convex*, or not. While convex optimization was arguably (and incorrectly) considered solved in the 1980s (Jordan, 2018), non-convex optimization had been considered difficult. However, with revived interest in training (deep) neural networks whose training loss functions virtually always fail to be convex, non-convex optimization has gained prominence. Perhaps surprisingly, many techniques from convex optimization such as the simple stochastic gradient descent (SGD) or accelerated methods (Nesterov, 1983) have proven to be effective for training non-convex neural networks as well.

Chapter 5 casts the program synthesis search problem as a supervised learning task, and shows that it can be effectively tackled by training a neural network via standard (non-convex) gradient-based optimization.

Bayesian optimization Although Bayesian optimization (Brochu et al., 2010; Shahriari et al., 2015) is a *method* for both discrete and continuous optimization problems, we mention it here separately thanks to its focus on optimizing expensive-to-evaluate black-box functions, often without access to its derivatives. In the discrete case this corresponds to the *best arm identification problem* (or *pure exploration* setting) in *Multi-Armed Bandits* (MAB, Bubeck et al., 2009).

Chen and Ghahramani (2016) showed how to convert a large-scale sampling problem into such a MAB problem using the *Gumbel trick*, and we also use this large-scale setup in Section 2.5.2 to evaluate our new methods for partition function estimation.

1.2.2 Integration

In this thesis, an integration problem is the task of evaluating a *definite* integral

$$I := \int_A f(x)\,\mathrm{d}\mu(x) \tag{1.3}$$

of a real-valued function $f : A \to \mathbb{R}$ with respect to some measure μ on A. This includes summation problems $\sum_{x \in A} f(x)$ on discrete domains A by taking μ to be the counting measure on A. However, note that this is different from *symbolic* integration, as we do not require that f has a closed-form anti-derivative.

Two primary sources of integration problems in machine learning are the needs of (1) computing normalizing constants of unnormalized distributions, and (2) evaluating expectations with respect to specific distributions.

Example 1 (Normalizing constants)**.** In an undirected graphical model (see also Section 1.5.1), the joint probability distribution is specified by a potential (negative energy) function $\phi : \mathcal{X} \to [-\infty, \infty)$ through $p(\mathbf{x}) \propto \exp(\phi(\mathbf{x}))$. Computing actual probabilities requires obtaining the normalizing constant $Z := \int_{\mathbf{x}} \exp(\phi(\mathbf{x})) \, \mathrm{d}\mathbf{x}$, which is also called the *partition function* in this context and its estimation will be the focus of Chapter 2.

Example 2 (Evaluating expectations)**.** After computing a Bayesian posterior of model parameters $p(\boldsymbol{\theta}|\mathbf{x})$ given observed data $\mathbf{x}$, one may wish to report the findings by providing scalar summaries, such as expected values of $\boldsymbol{\theta}$-dependent quantities (perhaps $\mathbb{E}[\boldsymbol{\theta}]$ itself).

Example 3 (Entropy estimation)**.** An important property of any probability distribution is the amount of uncertainty it encodes, which can be quantified by various measures of *entropy*. The Shannon entropy of a discrete distribution p is given by $H_1(p) := -\sum_{\mathbf{x}} p(\mathbf{x}) \ln p(\mathbf{x})$ and the Rényi entropy with parameter $\alpha \in (0, 1) \cup (1, \infty)$ is given by $H_\alpha(p) := \frac{1}{1-\alpha} \ln \sum_{\mathbf{x}} p(\mathbf{x})^\alpha$. Note that the former can be written as the expectation $\mathbb{E}_p[-\ln p(\mathbf{x})]$ with respect to the distribution p. The Rényi entropy can be expressed as a transformed expectation $H_\alpha(p) = \frac{1}{1-\alpha} \ln \mathbb{E}_p[p(\mathbf{x})^{\alpha-1}]$, which will be exploited in Chapter 3. See also Section 1.5.2 for important special cases of Rényi entropies.

Perhaps the most common type of integration problem in Bayesian machine learning is in fact both a normalizing constant evaluation problem, as well as an expectation evaluation problem. After application of Bayes theorem to compute a posterior distribution of interest,

$$p(\boldsymbol{\theta}|\mathbf{x}) = \frac{p(\mathbf{x}, \boldsymbol{\theta})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}}, \tag{1.4}$$

the denominator (called *model evidence*) is both the normalizer of the joint distribution $p(\mathbf{x}, \boldsymbol{\theta})$ in the numerator, as well as the integral of the likelihood $p(\mathbf{x}|\boldsymbol{\theta})$ with respect to the prior $p(\boldsymbol{\theta})$.

The model evidence is useful for (Bayesian) model selection when multiple models are under consideration. In particular, the ratio between two model evidences is called a *Bayes factor* and can be used to compare two competing model hypotheses. The term *marginal likelihood* is sometimes also used to refer to the model evidence, especially when $\boldsymbol{\theta}$ are latent (nuisance) variables, to be marginalized out. Optimizing the marginal likelihood across a space of models indexed by continuous model parameters is sometimes called model learning.

More generally, any *marginalization* problem of computing a distribution of a set of variables y_A from a joint distribution of y_A and y_B, leads to this type of integration problem. Similarly, any type of probabilistic conditioning leads to such a marginalization and thus integration problem through the definition of conditional probability.

Example 4 (Counting problems)**.** In a counting problem one wishes to count the number of elements in a larger set Ω satisfying a given predicate $P : \Omega \to \{\text{true}, \text{false}\}$. This can also be seen as an integration problem by integrating the indicator function $\mathbb{1}_P$ of the predicate with respect to the counting measure on Ω. Valiant (1979) defined the #P computational complexity class, comprising of counting problems where the predicate P can be evaluated in polynomial time. For example, the problem of counting the number of satisfying assignments to a given propositional formula is in #P. Note that counting problems in #P are at least as hard as their corresponding decision problems in NP, because it suffices to check whether the count is non-zero in order to solve the decision problem. In fact, the class #P is believed (but not proved) to be strictly harder than NP.

1.2.3 Sampling

A widespread use of sampling in machine learning is to perform Monte Carlo estimation of expectations, which is in fact a prime example of a conversion from an *integration* problem to *sampling* and will be discussed in Section 1.4.2. However, there are interesting and important cases where the randomness provided by *sampling* is useful in its own right; we list several relevant examples here.

Stochastic gradient methods In supervised machine learning one tends to minimize the (regularized) empirical risk on the training data, and if the empirical risk is differentiable – as it is with, say, most neural networks – gradient-based optimization methods are employed. However, instead of utilizing the exact gradient computed across the full training data, often one only considers a noisy version of it in each iteration by subsampling the training data (*mini-batch gradient descent*); in the extreme case of *stochastic gradient descent* (Robbins and Monro, 1951) the gradient is computed on a single, randomly chosen training data point.

However, the benefit of stochastic gradient methods lies not only in the computational cheapness of computing mini-batch gradients; there are also theoretical reasons to prefer them over full-batch methods (Bottou et al., 2018). Moreover, the noise present in the stochastic gradient is useful for finding minima of neural network loss landscapes that have better generalization properties (Dziugaite and Roy, 2017; LeCun et al., 2012).

In the i.i.d. setting this sampling problem is straightforward computationally, as one only needs to sample uniformly from a set of tractable size (the number of data points).

In Chapter 5 we use a more sophisticated variant of SGD called *Adam* (Kingma and Ba, 2015) to perform gradient-based optimization of a neural network in order to speed up a discrete search problem.

Differential privacy As machine learning solutions permeate everyday life, the question of individual privacy is gaining importance. On one hand, it has been shown that one can extract private information about training data points from a trained neural network (Fredrikson et al., 2015). On the other hand, in areas such as medicine strict regulations governing the use of individual data are already in place, but the ability to (safely) apply machine learning approaches could lead to substantial leaps in healthcare; e.g., better personalized treatment or cohort identification for clinical trials (Dankar and El Emam, 2013).

Differential privacy (Dwork, 2006) has emerged as the leading formalization of privacy that allows individuals to share their data in a way that allows quantifying how much privacy they lose by doing so. The main idea behind differentially private algorithms is that of addition of *noise*, which provides a form of *plausible deniability* to the participating individuals (Dwork and Roth, 2014), as illustrated by the following cartoon example.

Example 5 (Randomized response)**.** Suppose you want to find out the proportion of people in a conference room that have tried narcotics in their life. Asking those who have to raise their hand may underestimate the true value due to reluctance towards admitting such behaviour. However, as the conductor of the experiment you could first ask each person to discreetly throw a coin twice and remember the two results. If the first throw landed heads, they should answer your question truthfully, and otherwise they should randomize their answer according to the second coin toss. This way, anyone raising their hand now has *plausible deniability*, but the fraction of raised hands can still be used to estimate the true ratio with useful accuracy (Dwork and Roth, 2014). Moreover, by replacing fair coin tosses with biased ones it is possible to trade-off between the accuracy of this estimate and the privacy violation incurred by each participant.

In Chapter 4 we consider the problem of publicly releasing an entire database containing sensitive data, also known as *offline* differential privacy. To protect the privacy of the data, the released dataset will be a noisy version of the original database. A canonical algorithm for computing such a noised database, called *SmallDB* (Blum et al., 2008), requires sampling a synthetic database from a specific distribution supported on the set of all possible databases (with the same schema and size), which is computationally intractable for all but the simplest cases. Chapter 4 shows how the problem can be approached from a perspective of Reproducing Kernel Hilbert Spaces and then converted into an *optimization* problem.

Exploration algorithms In *bandit optimization* (Bubeck et al., 2009) and *reinforcement learning* (RL, Sutton and Barto, 1998) an agent interacts with an environment and receives rewards. Since the distribution of rewards (and in the case of RL also the dynamics of the environment) are unknown to the agent, it needs to *explore* the environment to discover areas with high reward. Randomized exploration strategies have been shown to be effective for achieving this goal. A well-known example is Thompson sampling (Thompson, 1933) where one point is drawn from the distribution encoding current beliefs of the agent about its environment, and the next action is optimized with respect to this one sample. Thompson sampling also forms basis of posterior sampling reinforcement learning (PSRL, Dearden et al., 1998), an exploration strategy in RL.

Bandit optimization problems can also arise artificially; for example, Chen and Ghahramani (2016) cast a large-scale discrete sampling problem as a bandit optimization

problem using the *Gumbel trick*. We adapt this setup to partition function estimation and use it to evaluate our new generalizations of the Gumbel trick in Section 2.5.2.

Other randomized algorithms Sampling is generally required in all *randomized algorithms*. A specific problem type where randomized algorithms can be useful are *adversarial* setups, where an online algorithm operates against an adaptive adversary whose actions can depend on previous decisions of the algorithm. Intuitively, randomization is helpful because the adversary at least cannot know the next action of the algorithm. For example, the *paging problem* is the problem of deciding which documents to keep in a fast cache of limited size in order to minimize the total number of future cache misses. Fiat et al. (1991) showed that a randomized algorithm can achieve a strictly better competitive ratio against the best possible offline algorithm than any deterministic algorithm could.

1.2.4 Search

Classical AI "Classical" artificial intelligence (Russell and Norvig, 2016) as pursued since the 1950s has focused on techniques using explicit knowledge representations, logical reasoning, and planning. Examples of common problem formalizations were graph problems, constraint satisfaction problems (CSPs), and games. These problems were addressed through discrete or continuous search algorithms, a basic but well-known example of which is A^* search (Hart et al., 1968).

Program synthesis Even as researchers aspiring to development of artificial intelligence increasingly look towards statistical techniques, search problems are still present in the machine learning domain. In fact, some believe that a key stepping stone towards building intelligent agents can be achieved through progress in *program synthesis* (Goodman et al., 2014, Section 7), and indeed program synthesis has been considered a "holy grail of computer science" (Gulwani et al., 2017). In program synthesis the task is to automatically construct program source code from a given specification, such as input-output examples. The high-level idea is that an agent released in an unknown environment should be able to learn the laws governing said environment from interactions (inputs) and observations (outputs) in order to be able to plan its future actions (Reed and De Freitas, 2016). While a similar goal is pursued by the related field of *program induction* (Graves et al., 2014), where a neural network

learns a program implicitly in its trainable weights, learning an explicit program has several advantages:

1. a restricted programming language or a preference for shorter programs can encode the useful *inductive bias* towards simpler laws of the world,
2. a program is *interpretable* by a human and can be *assessed* for correctness, and
3. it can be further *modified* and *composed* in transparent ways.

In fact, in recent years there has been renewed interest in combining symbolic reasoning approaches with learning based systems in a symbiotic manner. An early example of this type of modern *neuro-symbolic* approach is the work on guiding program synthesis search using a trained neural network, which will be presented in Chapter 5.

Automated theorem proving A related area to program synthesis is that of *automated theorem proving*, where the task is to automatically construct proofs of a given mathematical statement by suitably combining a (large) set of given axioms and lemmas. Similarly to program synthesis, approaches that combine symbolic search with neural networks have been pursued (Alemi et al., 2016).

1.2.5 Other Fundamental Problems

There are other abstract tasks beyond optimization, integration, sampling, and search that a machine learning algorithm might need to carry out. However, these tend to be either easier or less common. The most prominent example is the task of computing a derivative at a given point (*differentiation*), for which the Chain rule and its more sophisticated use in the *backpropagation* algorithm (Kelley, 1960) provide an efficient and deterministic solution that covers most use cases. Other examples of tasks might include symbolic differentiation and symbolic integration, or solving differential equations (Lample and Charton, 2020).

1.3 Conversions

Reductions and conversions Theoretical computer science and in particular the study of formal models of computation and complexity theory rely heavily on the notion of *reductions*, which are *computable* functions (algorithms) that transform one

problem into another. A reduction f from a problem A to another problem B is useful because it shows that the problem A is computationally no harder than the combined computational complexity of problem B and of computing the reduction f. For example, a *Turing reduction* (an arbitrary computable function) from an undecidable problem A to another problem B proves that the latter is also undecidable. Similarly, a *polynomial-time reduction* from an NP-hard problem A to another problem B proves that the latter is also NP-hard.

This thesis focuses on problem transformations that translate between different types of abstract fundamental problems discussed in the preceding Section 1.2. However, they are not necessarily *reductions* in the formal sense, because in some cases the application permits that the solution after applying the transformation is only approximate, or that solving the transformed problem requires an additional resource type (such as data). Hence, to avoid confusion, in this thesis we will use the term *conversion* (instead of reduction) to describe the problem transformations considered. However, they do share with formal reductions the notion of efficient computability, as we seek conversions that are practically applicable.

Why conversions? A natural question to ask is why one should consider converting between different problem types at all, if one is interested in solving them rather than proving hardness results. Why is it not simpler to construct a solution within the original problem type? Indeed, as expected we shall see that none of the existing conversions discussed later in Section 1.4 are a silver bullet that would provide a general performance gain on its own. However, the following list shows broad categories of situations where a conversion to a different problem type can be beneficial (the first three points are closely related to each other). After the high-level list, the specific examples of the four conversions studied in this thesis are categorized. Section 1.4 then provides more examples of existing useful conversions within the field of machine learning.

- *Historical development*
  Although all of *optimization*, *integration*, *sampling* and *search* are well-studied problems, none is considered completely solved, and researchers working on each of these problems have had different perspectives, motivations, and backgrounds, leading to some sub-types of these problem classes being better explored than others. As a result, converting into a different problem type might map onto a specific problem instance that has been more thoroughly studied.

- *Well-studied problem structure*
  The Gumbel trick and its generalizations presented in Chapters 2 and 3 not only map onto the well-studied problem of finding the most likely configuration in a discrete graphical model (an example of the previous point), the conversion is such that it preserves submodularity properties of the discrete graphical model, and as such makes it amenable to fast MAP solvers in common cases.

- *Effectiveness of heuristics*
  The previous two points are especially relevant for heuristics, as different fields have developed their own set of heuristics following their applications of interest.

- *Unlocking new trade-offs*
  When designing machine learning solutions, one needs to take multiple types of costs and constraints into account. Apart from the typical notions of (1) computation time and (2) space (memory) that are ubiquitous in computer science, these are also (3) sample (data) complexity, (4) statistical risk, (5) privacy, (6) fairness, and possibly others. Converting to a different problem type can unlock different types of trade-offs between these quantities.

- *Implementation advantages and convenience*
  Converting into a different problem type could also be done for convenience or availability of robust implementations. For example, when wishing to sample from a trained RNN decoder outputting logits (unnormalized log probabilities), a convenient way is to utilize the Gumbel trick, which in this case has the same time and memory complexity as standard sampling but it is simpler to implement and numerically more stable, because it does not require normalization of probabilities and instead acts directly on the logits.

- *New understanding*
  Finally, converting to a different problem type can lead to a better understanding of the original problem. For example, rephrasing a problem as an optimization problem using duality or the variational framework more generally, one can consider perturbations of the derived optimization problem and study how they affect the resulting solution (Jordan et al., 1999).

The conversions studied in this thesis tend to fall into multiple categories:

- The Gumbel trick (Papandreou and Yuille, 2011) and its generalizations presented in Chapters 2 and 3 convert integration problems to the well-studied optimization

problem of finding the MAP (most likely) configuration in a discrete graphical model. However, they also introduce a new type of trade-off between computation time and statistical accuracy, as they lead to stochastic approximation schemes.

- The work on differentially private database release via kernel mean embeddings presented in Chapter 4 yields an optimization algorithm with the *anytime* property and thus introduces a new trade-off between computation time and accuracy. The original *sampling* problem would have a trade-off between computation time and privacy, which would be useless if one wants to guarantee that a certain privacy level is attained (which is indeed the point of differential privacy). Moreover, the conversion allows utilizing any optimization heuristics on the generated *reduced set* optimization problem (Schölkopf and Smola, 2002, Chapter 18).

- In Chapter 5, casting the discrete search in program synthesis as a supervised learning task that can be solved using neural networks unlocks the ability to exploit the efficiency of gradient-based optimization of neural network models (which one might consider a heuristic for minimizing non-convex loss functions) in order to solve an inherently discrete problem.

1.4 Examples of Conversions in Machine Learning

1.4.1 Conversions to Optimization Problems

Variational inference *Variational methods*, historically derived from the field of *calculus of variations* (Lagrange, 1788), have entered machine learning through *mean field methods* from statistical physics (Parisi, 1988). Variational methods have turned out to be very successful within the formalism of graphical models (Jordan et al., 1999; Wainwright and Jordan, 2008), where they provide a powerful paradigm for approximate *(variational) inference* by rewriting an *integration* problem into an *optimization* problem.

In modern probabilistic machine learning one often posits a generative model involving observed variables $\mathbf{x}$ (the *data*), latent variables $\mathbf{z}$ (to be *inferred*), and model parameters $\boldsymbol{\theta}$ (to be *learned*). The generative distribution can be described by a directed graphical model, with edges indicating probabilistic dependencies (conditional probability distributions). With such a probabilistic formulation it is natural to take, at least partially, a Bayesian approach and work with posterior distributions $p(\cdot|\mathbf{x})$, such as the joint posterior of all unobserved variables $p(\mathbf{z}, \boldsymbol{\theta}|\mathbf{x})$. However,

commonly required operations such as *sampling*, marginalization, and evaluating expectations (both examples of *integration* problems) under posterior distributions can be computationally intractable, especially when there is a large number of unobserved variables (Zhang et al., 2018).

Variational inference (Blei et al., 2017) takes the approach of defining a *variational family* of probability distributions $\mathcal{Q} = \{q_{\boldsymbol{\lambda}} : \boldsymbol{\lambda} \in \Lambda\}$, parametrized by *variational parameters* $\boldsymbol{\lambda}$, such that (1) required operations are tractable for distributions in $\mathcal{Q}$, (2) $\mathcal{Q}$ is rich enough to approximate the posterior distribution of interest sufficiently well, and (3) optimization over $\mathcal{Q}$ (or Λ) is tractable. The algorithmic crux of variational inference is then to solve the *optimization* problem

$$q^* \leftarrow \operatorname*{argmin}_{q \in \mathcal{Q}} D(p, q) = q_{\operatorname{argmin}_{\boldsymbol{\lambda} \in \Lambda} D(p, q_{\boldsymbol{\lambda}})}, \tag{1.5}$$

where p is the posterior of interest, and $D(p, q)$ is a divergence measure between distributions p and q (a non-negative function that equals 0 if and only if $p = q$). The most common choice is the *reverse* Kullback-Leibler (KL) divergence $D(p, q) = \mathrm{KL}(q \| p) = \int q \ln \frac{p}{q}$, although other choices including Rényi α-divergences have also been explored (Li and Turner, 2016).

Graph message passing algorithms *(Loopy) belief propagation* (Murphy et al., 1999; Pearl, 1988) and *Expectation maximization* (EP) (Minka, 2001) are two examples of approximate inference methods for graphical models that can be interpreted on one hand as (1) message passing algorithms on the graph, but also as (2) solving an optimization problem (although with EP the situation is more subtle, Bui et al., 2016). Wainwright and Jordan (2008) thus argued that these methods should be seen as part of the variational methods family, because they involve an optimization to approximate a density of interest.

Perturb-and-MAP *Perturb-and-MAP* methods (Papandreou and Yuille, 2011) constitute a class of randomized algorithms for converting the *sampling* problem, and the problem of estimating normalizing constants (an *integration* problem) into the *optimization* problem of finding a highest-probability configuration. The algorithms operate by first randomly perturbing the target probability distribution and then employing maximum a posteriori (MAP) solvers on the perturbed models to find a maximum probability configuration, hence the name of the framework. As the

random perturbations involve the Gumbel distribution (Gumbel, 1935), the framework is sometimes also described as (utilizing) the *Gumbel trick*.

More specifically, Papandreou and Yuille (2011) first introduced the framework in the context of discrete graphical models. Given a potential function $\phi : \mathcal{X} \to [-\infty, \infty)$ defining the *Gibbs distribution* $p(\mathbf{x}) \propto \exp(\phi(\mathbf{x}))$ on a finite state space $\mathcal{X}$, the problem of *sampling* p can be approached by adding an independent Gumbel perturbation to each configuration's potential, and then finding the highest-potential configuration. Hazan and Jaakkola (2012) pointed out that such perturbations are useful not only for sampling the Gibbs distribution (considering the *argmax* of the perturbed model), but also for bounding and approximating the normalizing constant (by considering the value of the *max*), which is also known as the *partition function* in this context. Later, Maji et al. (2014) exploited the duality relationship between the partition function and the Shannon entropy to construct perturb-and-MAP estimators of this entropy.

Although the resulting MAP problems are NP-hard in general, substantial research effort has led to the development of solvers which can efficiently compute or estimate the MAP solution on many problems that occur in practice (e.g., Boykov et al., 2001; Darbon, 2009; Kolmogorov, 2006). Evaluating the partition function is a harder problem, containing #P-hard counting problems.

Specific instances of the Perturb-and-MAP framework and their applications are described in more detail in Section 2.1. Chapter 2 also introduces a generalization of the *Gumbel trick* to a wider family of algorithms utilizing different perturbation distributions, some of which have more favourable statistical properties when estimating normalizing constants. Chapter 3 presents an extension of the Gumbel trick (and also of the new tricks from Chapter 2) to a new *integration* problem, the problem of estimating Rényi entropies.

Other partition function estimators via MAP calls Other methods that estimate the partition function of a discrete graphical model via MAP solver calls include:

- The WISH method (weighted-integrals-and-sums-by-hashing, Ermon et al., 2013) relies on repeated MAP inference calls applied to the model after subjecting it to random hash constraints.
- The Frank-Wolfe method may be applied by iteratively updating marginals using a constrained MAP solver and line search (Belanger et al., 2013; Krishnan et al., 2015).

- Weller and Jebara (2014a) used a single MAP call over a discretized mesh of marginals to approximate the Bethe partition function, which itself is an estimate of the true partition function.

Search by optimization Suppose one wants to find an element $x \in \mathcal{X}$ on a (finite or infinite) search space $\mathcal{X}$ satisfying a predicate $P : \Omega \to \{\text{true}, \text{false}\}$. One can trivially set up a maximization *optimization* problem $x^* \leftarrow \operatorname{argmax}_{x \in \mathcal{X}} \mathbb{1}_P(x)$ that is computationally equivalent to the original search problem. Naturally, such a reformulation is not useful on its own, but can be made such through a relaxation – for example, replacing $\mathbb{1}_P(x)$ with a smoother (e.g., differentiable) function $Q : \mathcal{X} \to [0, 1]$ that approximates $\mathbb{1}_P$. The work presented in Chapter 5 can be seen as an instance of this general approach, because a neural network is trained to make probabilistic predictions about properties of the solution that is being searched for, and thus induces a function Q on the search space of programs $\mathcal{X}$ that approximates the indicator $\mathbb{1}_P$ of the predicate P.

Another type of relaxation is when the space $\mathcal{X}$ itself is extended, such as when $\mathcal{X}$ is discrete but Q can be defined on a larger $\mathcal{X}' \supsetneq \mathcal{X}$ such that $Q(x)$ is close to $\mathbb{1}_P(x)$ on $\mathcal{X}$, and Q is continuous on $\mathcal{X}'$. An instance of this is a search problem encoded as an integer programming problem, which is in turn relaxed to linear programming by elimination of integrality constraints on the variables.

1.4.2 Conversions in Other Directions

Although not a focus of this thesis, for completeness we provide three important examples of conversions between fundamental problems that do not reduce to an optimization problem, but instead go in different directions.

Integration to sampling Perhaps the most widely known conversion technique of all is that of Monte Carlo sampling, where the *integration* problem of evaluating a definite integral $I := \int f(x)p(x)\,\mathrm{d}x$ with respect to a probability density $p(x)$ is converted into the problem of *sampling* points $x_1, \ldots, x_N$ from p or an approximation thereof such that the quantity of interest can be approximated by $\hat{I} := \frac{1}{N}\sum_{n=1}^{N} f(x_n)$.

Optimization to sampling Suppose one wishes to minimize a function $f : \mathcal{X} \to \mathbb{R}$. One can set up the problem of *sampling* a probability distribution p on $\mathcal{X}$ with density $p_\beta(x)$ proportional to $\exp(-\beta f(x))$, where $\beta > 0$ is an inverse temperature parameter.

As β is annealed to ∞, samples from $p_\beta(x)$ will concentrate towards minima of f. This classical technique is called *simulated annealing*.

Sampling to integration Sampling a structured probability distribution $p(x_1, \ldots, x_n)$ on $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, such as a discrete graphical model, can be performed much faster if one can efficiently solve *integration* problems of evaluating marginals (and conditionals). Specifically, one can sample X_1 from the marginal distribution of $p(x_1)$ and then for each $i = 2, \ldots, n$ sample X_i from the conditional distribution $p(x_i \mid x_1, \ldots, x_{i-1})$. This brings down the computational complexity of sampling from $\mathcal{O}(|\mathcal{X}|)$ (exponential in n) to $\mathcal{O}(n \max_i |\mathcal{X}_i|)$ (linear in n). A conversion of this type is used by Hazan et al. (2013) in the Perturb-and-MAP framework, and also by us in Section 2.3.3.

1.5 Tasks Considered in This Thesis

This section introduces the specific tasks for which subsequent chapters of the thesis present beneficial conversions to optimization problems.

1.5.1 Estimating Normalizing Constants

Given an unnormalized probability distribution, i.e. an unnormalized probability mass function in the discrete case or an unnormalized density function in the continuous case, denoted by $\tilde{p} : \mathcal{X} \to [0, \infty)$, the task is to compute the normalization constant $Z := \int_{\mathcal{X}} \tilde{p}$ (which corresponds to the summation $Z := \sum_{x \in \mathcal{X}} \tilde{p}(x)$ in the discrete case by considering the counting measure on $\mathcal{X}$). Section 1.2.2 outlined how this *integration* problem commonly arises in machine learning applications involving undirected graphical models, Bayesian modelling, and marginalization or conditioning of probability distributions more generally; these are all fundamental tools with ubiquitous applications.

Undirected graphical models The case of estimating normalizing constants of an undirected graphical model will be of particular interest in Chapter 2 and Section 2.3 more specifically. Given an undirected graph $G = (V, E)$ where the vertices V index random variables $\mathbf{X} = \{X_v \mid v \in V\}$ taking values in $\mathcal{X} = \prod_{v \in V} \mathcal{X}_v$, an *undirected graphical model*, also known as a *Markov random field*, is a probability distribution

over $\mathcal{X}$ that factorizes along the cliques (maximal fully-connected subgraphs) of G:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{\substack{C \subseteq V \\ \text{a clique}}} \psi_C(\mathbf{x}_C), \qquad (\mathbf{x} \in \mathcal{X}). \tag{1.6}$$

Here $\mathbf{x}_C := \{x_v \mid v \in C\}$ and $\psi_C : \prod_{v \in C} \mathcal{X}_v \to [0, \infty)$ describes the mutual compatibility of assignments to random variables in the clique C. In this context, the scalar normalizing constant Z is called the *partition function*. The term comes from statistical physics, where the partition function describes how probability mass is partitioned among possible microstates of a system depending on their energies. The non-negative compatibility functions ψ_C can always be rewritten in exponential form to reveal what is called a *Gibbs distribution*, or in finite-variable cases also the *Boltzmann distribution*:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{\substack{C \subseteq V \\ \text{a clique}}} \psi_C(\mathbf{x}_C) = \frac{1}{Z} \exp\left(\sum_{\substack{C \subseteq V \\ \text{a clique}}} \phi_C(\mathbf{x}_C) \right) = \frac{1}{Z} \exp\left(- \sum_{\substack{C \subseteq V \\ \text{a clique}}} E_C(\mathbf{x}_C) \right). \tag{1.7}$$

Here ϕ is called the *potential* function, and E the *energy*; they are related simply by a sign change. Chapters 2 and 3 will be phrased in terms of the potential function $\phi(\mathbf{x}) := \sum_C \phi_C(\mathbf{x}_C)$, which one can recognize as simply the logarithm of the unnormalized probability mass or density function: $\phi(\mathbf{x}) = \ln(Zp(\mathbf{x}))$.

Discrete graphical models The framework of undirected graphical models permits both discrete and continuous random variables, or indeed a combination thereof. Although the continuous case will be considered in Section 2.5.1, our focus will be on the setting where all random variables X_i take values in finite sets $\mathcal{X}_i$, in which case we talk about *discrete graphical models*.

A canonical example of a discrete graphical model is an *Ising model* (Lenz, 1920), again from statistical physics. It models a system of magnetic atoms, where each can have a positive or negative spin: the variable X_i represents the spin of atom i and takes values in $\{-1, +1\}$. Edges of the graphical model link adjacent atoms, encoding that there is an interaction between them. In the case of an Ising model the edges usually form a 2D or 3D lattice, and consequently all cliques consist of just two vertices joined by an edge. The potential function thus consists of *pairwise* potentials: $\phi(\mathbf{x}) = \sum_{e \in E} \phi_e(\mathbf{x}_e)$. Extensions include the addition of an external magnetic field, which introduces *unary* potentials $\phi_v(x_v)$ into the potential function,

and the generalization to more than two spins, which is then called a *Potts model* (Potts, 1952).

Apart from widespread use in statistical physics, applications of discrete graphical models in machine learning include Hopfield networks (Little, 1974) for simulating an associative memory, and the general class of Boltzmann machines (Ackley et al., 1985) including Restricted Boltzmann machines (RBM, Smolensky, 1986) and Deep Boltzmann machines (DBM, Salakhutdinov and Hinton, 2009) whose addition of hidden units (with values not observed during training) permits representation learning and have been used for tasks such as collaborative filtering (Salakhutdinov et al., 2007), classification (Larochelle and Bengio, 2008) and sequence modelling (Sutskever and Hinton, 2007).

Computing the partition function of a discrete graphical model is in general equivalent to computing a matrix permanent, which is a #P-hard problem (Valiant, 1979), a computational complexity class that is believed to be even harder than NP.

1.5.2 Rényi Entropy Estimation

Entropies are measures of uncertainty in a probability distribution. The most commonly used is the Shannon entropy (Shannon, 1948), defined by $H_1(\mathbf{p}) := \sum_{x\in\mathcal{X}} p(x) \ln p(x)$. The Shannon entropy possesses strong information-theoretic properties and its estimation has found a wide array of applications within statistics (Goria et al., 2005), machine learning (Learned-Miller and John III, 2003), as well as other fields (Porta et al., 2001).

The Rényi entropy (Rényi, 1960) can be seen as a generalization of the Shannon entropy into a continuous class of entropies parametrized by $\alpha \in [0, \infty]$. For $\alpha \in (0, 1) \cup (1, \infty)$ the α-Rényi entropy is given by $H_\alpha(\mathbf{p}) := \frac{1}{1-\alpha} \ln \sum_{x\in\mathcal{X}} p(x)^\alpha$, and it is extended to the boundary values via appropriate limits. This definition encompasses and generalizes multiple measures of uncertainty with important roles in different fields:

- $\alpha = 0$ is the *Hartley entropy*, related to the distribution's support size;
- $\alpha = 1$ is the *Shannon entropy*, the most widely used entropy measure;
- $\alpha = 2$ is the *collision entropy*, equal to the negative logarithm of the probability that two independent draws from the distribution coincide;
- $\alpha = \infty$ is the *min-entropy*.

Section 3.2.1 provides more information on particular α-Rényi entropies.

Literature on entropy estimation usually assumes access to N samples $x_1, \ldots, x_N \overset{\text{iid}}{\sim} p$ from the unknown distribution p, and the task is to estimate the entropy of p from these N points. In the perturb-and-MAP framework no samples from p are assumed; instead, the distribution of interest is described by a potential function $\phi : \mathcal{X} \to [-\infty, \infty)$, as in Section 1.5.1. Computing the entropy remains a challenging problem in this setting because it amounts to computing an integral over the support of p. Maji et al. (2014) exploited the duality relationship between the partition function and the Shannon entropy to construct a perturb-and-MAP estimator of the Shannon entropy based on the Gumbel trick. Chapter 3 shows how the perturb-and-MAP framework (including both the Gumbel trick and the new tricks from Chapter 2) can be cleanly extended to the integration problem of estimating Rényi entropies. The relationship with the existing estimator of the Shannon entropy due to Maji et al. (2014) is also discussed in detail.

1.5.3 Differentially Private Database Release

In Chapter 4 we are concerned with the following problem. Given a database of records containing sensitive data (e.g., patient health records), the trustworthy curator of this database wishes to release (some form of) this data to the public, so that

1. useful statistical inferences can be drawn from this data (e.g., relationships between patient features and treatment outcomes), and

2. the data can be combined with other sources in order to unlock the potential of large-scale machine learning approaches (e.g., combining health data from different hospitals, regions, or countries).

This setting is sometimes called *offline*, or *non-interactive* differential privacy (Blum et al., 2008; Dwork and Roth, 2014), because at the time of release it is not known what kinds of statistical algorithms the public might wish to execute on the released data. As a result, it is desirable to impose as few restriction as possible, and release a version of the data that can be used to answer as large a diversity of queries as accurately as possible, as long as the definition of differential privacy is satisfied.

This setup is different from most algorithms studied in differential privacy, which are concerned with safely releasing a particular statistic of the sensitive data (e.g., the average glucose level among patients), or a particular statistical model trained

on the sensitive data points (e.g., the weights of a linear regression model mapping blood readings to life expectancy) (Chaudhuri et al., 2011; Rubinstein et al., 2012). Intuitively, the offline setting is more difficult because the privatization step cannot be specialized to the end-to-end application, but instead needs to support a variety of use cases (Kowalczyk et al., 2018).

As discussed in Section 1.2.3, differential privacy is achievable by addition of suitable noise during the release process. Chapter 4 shows how the intractable sampling problem arising in the canonical offline differential privacy algorithm *SmallDB* (Blum et al., 2008) can be replaced with the optimization problem of minimizing a distance in a Reproducing Kernel Hilbert Space (RKHS, Smola et al., 2007). The output of these algorithms is a noisy version of the original database, thus sometimes called a *synthetic* database. It can be safely released to the public, because the *post-processing* theorem of differential privacy (Dwork and Roth, 2014) ensures that no further computation (without access to the private data) can undo the privacy guarantee of an object that had been released in a differentially private manner.

1.5.4 Program Synthesis from Input-Output Examples

Program synthesis (Gulwani et al., 2017) is the task of synthesizing program source code from a given specification. A prominent type of specification is a set of input-output examples, in which case we can talk more specifically about *programming by example* (Cypher and Halbert, 1993). Another kind of specification is a natural language description of the desired functionality, in which case the task can be seen as an instance of *semantic parsing* (Liang, 2016).

A successfully deployed instance of programming by example is *FlashFill* (Gulwani, 2011; Gulwani et al., 2012), which has shipped as a feature of Microsoft Excel permitting users to obtain string-manipulation macros just by providing input-output examples in two adjacent columns of the spreadsheet. For instance, if the first column of a spreadsheet contains a list of person full names, and the user starts filling in the second column with surnames, FlashFill can automatically synthesize a macro that performs the extraction (assuming, say, that the surname is always the last token of the full name) and offer to fill in the rest of the second column using this macro. Note that the macro will be visible to the user, who can check it, modify it, and even learn from it in order to improve their own macro writing abilities.

As program synthesis systems continue evolving, they hold promise at different levels of application:

1. First, they can usefully augment the "intelligence" of trained software engineers by synthesizing repetitive or conceptually simple API manipulations for them, thus increasing their productivity. An example of this would be *AutoPandas* (Bavishi et al., 2019), a program synthesis system for the data science Python library *Pandas*, which has a notoriously wide API.

2. Second, they can open up the world of programming to non-experts by facilitating source code generation through them only providing input-output examples or another type of specification. An early example of this is the *FlashFill* system (Gulwani, 2011; Gulwani et al., 2012) described above.

3. Third, autonomous agents operating in unknown environments can formulate interpretable, generalizable, and compositional laws (in the form of computer programs) governing their environment from interaction observations using techniques from program synthesis (Goodman et al., 2014).

Chapter 5 presents a methodological advancement on the task of program synthesis from an input-output specification, showing how prior success in deep learning can be exploited in this domain. The discrete search problem of identifying a program matching the specification is cast as a supervised learning task that can be effectively attacked using deep neural networks and gradient-based optimization. Although we only consider synthesizing simple integer list-manipulation programs with basic arithmetic in this work, combining predictions of the trained neural network with existing symbolic search techniques already leads to orders of magnitude speed-ups in terms of wall clock time.

1.6 Contributions and Thesis Organization

In the abstract context of converting *non*-optimization machine learning tasks to optimization problems, the main contributions of this thesis are:

1. A generalization of an existing method to convert partition function estimation (an *integration* problem) into an optimization problem, which leads to novel conversions that achieve lower statistical error at the same computational cost. (Chapter 2)

2. An extension of the aforementioned methods to the *integration* problem of estimating Rényi entropies. (Chapter 3)

3. A novel algorithm for differentially private release of a database containing sensitive data, based on the theory of Reproducing Kernel Hilbert Spaces. Computationally, the resulting algorithm can be viewed as replacing an intractable *sampling* problem by an optimization procedure that can be stopped early, thus yielding a more practical *anytime* algorithm. (Chapter 4)

4. Learning Inductive Program Synthesis (LIPS), a formulation of program synthesis from input-output examples that allows exploiting gradient-based optimization for accelerating a discrete *search* task. (Chapter 5)

In all four cases, the contribution yields an algorithm with previously unattainable properties: in Chapter 2 this is up to 40% higher statistical efficiency, in Chapter 3 an *unbiased* Perturb-and-MAP estimator of Rényi entropies, in Chapter 4 a novel synthetic database algorithm with provable mathematical guarantees, and in Chapter 5 a computational speedup of 1-3 orders of magnitude. These instances corroborate the thesis that conversions between different abstract problem types can be beneficial for deriving useful algorithms and theoretical results.

The material presented in Chapters 2, 4 and 5 is based on work presented in the following peer-reviewed publications, respectively:

- **Lost Relatives of the Gumbel Trick**
  Matej Balog, Nilesh Tripuraneni, Zoubin Ghahramani, Adrian Weller
  ICML 2017, Sydney, Australia (Balog et al., 2017b)

- **Differentially Private Database Release via Kernel Mean Embeddings**
  Matej Balog, Ilya Tolstikhin, Bernhard Schölkopf
  ICML 2018, Stockholm, Sweden (Balog et al., 2018)

- **DeepCoder: Learning to Write Programs**
  Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, Daniel Tarlow
  ICLR 2017, Toulon, France (Balog et al., 2017a)

These publications resulted from very enjoyable collaborations with the listed co-authors. In all three cases, the author of this thesis was the only "first" author on the publication.

Thesis organization As the case studies presented in this thesis are linked by the abstract philosophical notion of converting non-optimization tasks to optimization problems rather than by a single common technique or application domain, more detailed background and related work beyond what was already discussed in Sections 1.4 and 1.5 is provided at the beginning of each respective chapter 2, 3, 4, and 5. Each chapter then presents its algorithmic contribution, with corresponding mathematical and/or experimental analysis. In an attempt to increase transparency of presentation, some technical details that can be abstracted away on first reading, such as technical proofs and details of experimental setups, are postponed into Appendices A, B, C, and D. The chapters conclude with discussions of future directions – both those that have already been undertaken since first publication, and those that still remain open.

1.7 Other PhD Work

During my PhD I've attempted to build a network of research collaborations, which has fortunately resulted in more scientific work that is not included in this thesis, primarily due to space limitations. Some of this work also has a less direct connection to the overarching theme of conversions to optimization problems, although all of them do share the aspect of setting up an (auxiliary) optimization problem in order to solve the target task of interest.

- **Neural grammar learning** (2017)
  with Alex Polozov and Rishabh Singh
  We were addressing the problem of discovering rules of a (formal) grammar that governs utterances in a given dataset. The target application was *fuzzing*: given a set of valid inputs for a program binary, can we discover a grammar describing valid inputs, so as to be able to generate new inputs and test that the binary does not crash on them?
  Similarly to DeepCoder, the work described in Chapter 5, we approached this *discrete search* problem by setting up a supervised learning task and used *gradient-based optimization* to learn a neural network that consumes a set of utterances and predicts the grammar rules in the form of a regular expression.

- **Fast training of sparse graph neural networks on dense hardware** (2019)
  with Bart van Merriënboer, Subhodeep Moitra, Yujia Li, Daniel Tarlow
  With sparse graph neural networks (GNNs) emerging as strong encoder models

for program source code (Allamanis et al., 2018), and the simultaneous rise in popularity of "AI accelerators" (such as Google's TPUs, Hennessy and Patterson, 2019) focused on fast multiplication of dense matrices, we explored the question whether committing to such hardware necessarily prevents accelerating models requiring sparse operations (such as a pre-multiplication by a large, sparse graph adjacency matrix as in the case of sparse GNNs). We approached the problem by thinking in terms of an abstract *virtual machine* that supports operations that are fast on "AI accelerators" (i.e. dense matrix multiplies) and designed a *compilation* step that expresses the sparse operations required by a sparse GNN in terms of operations supported by the virtual machine. The introduction of the compilation step can be seen as an addition of an auxiliary *optimization* task to speed up a target *optimization* task of interest.

- **Towards program synthesis in latent space** (2019)
  with Rishabh Singh, Petros Maniatis, Charles Sutton
  As discussed in detail in Section 5.7.1, the DeepCoder work left many avenues open for further improvement. In this project we have explored some of those, attempting to learn a latent space representation of computer programs that could be more tightly coupled with a search procedure looking for a program matching an input-output specification. This work continues in the spirit of DeepCoder to speed up a *discrete search* procedure through *gradient-based* optimization.

Chapter 2

Partition Function Estimation

Chapter abstract The Gumbel trick is a method to sample from a discrete probability distribution, or to estimate its normalizing partition function. The method relies on repeatedly applying a random perturbation to the distribution in a particular way, each time solving for the most likely configuration. We derive an entire family of related methods, of which the Gumbel trick is one member, and show that the new methods have superior properties in several settings with minimal additional computational cost. In particular, for the Gumbel trick to yield computational benefits for discrete graphical models, Gumbel perturbations on all configurations are typically replaced with so-called low-rank perturbations. We show how a subfamily of our new methods adapts to this setting, proving new upper and lower bounds on the log partition function and deriving a family of sequential samplers for the Gibbs distribution. Finally, we balance the discussion by showing how the simpler analytical form of the Gumbel trick enables additional theoretical results.

2.1 Introduction

In this chapter we are concerned with the fundamental problem of sampling from a discrete probability distribution and evaluating its normalizing constant, the setup introduced in Section 1.5.1: a probability distribution p on a discrete sample space $\mathcal{X}$ is provided in terms of its potential function $\phi : \mathcal{X} \to [-\infty, \infty)$, corresponding to log-unnormalized probabilities via $p(\mathbf{x}) = e^{\phi(\mathbf{x})}/Z$, where the normalizing constant Z is the *partition function*. In this context, p is the *Gibbs distribution* on $\mathcal{X}$ associated with the potential function ϕ. The challenges of sampling from such a discrete probability distribution and estimating the partition function are fundamental problems with

ubiquitous applications in machine learning, classical statistics and statistical physics (see, e.g., Lauritzen, 1996).

Perturb-and-MAP methods (Papandreou and Yuille, 2010) constitute a class of randomized algorithms for estimating partition functions and sampling from Gibbs distributions, which operate by randomly perturbing the corresponding potential functions and employing maximum a posteriori (MAP) solvers on the perturbed models to find a maximum probability configuration. This MAP problem is NP-hard in general; however, substantial research effort has led to the development of solvers which can efficiently compute or estimate the MAP solution on many problems that occur in practice (e.g., Boykov et al., 2001; Darbon, 2009; Kolmogorov, 2006). The general aim of perturb-and-MAP methods is to reduce the problem of partition function evaluation, or the problem of sampling from the Gibbs distribution, to repeated instances of the MAP problem (where each instance is on a different random perturbation of the original model).

The Gumbel trick (Papandreou and Yuille, 2011) relies on adding Gumbel-distributed noise to each configuration's potential $\phi(\mathbf{x})$. We derive a wider family of perturb-and-MAP methods that can be seen as perturbing the model in different ways – in particular using the Weibull and Fréchet distributions alongside the Gumbel. We show that the new methods can be implemented with essentially no additional computational cost by simply averaging existing Gumbel MAP perturbations in different spaces, and that they can lead to more accurate estimators of the partition function.

Evaluating or perturbing each configuration's potential with i.i.d. Gumbel noise can be computationally expensive. One way to mitigate this is to cleverly prune computation in regions where the maximum perturbed potential is unlikely to be found (Chen and Ghahramani, 2016; Maddison et al., 2014). Another approach exploits the product structure of the sample space in discrete graphical models, replacing i.i.d. Gumbel noise with a "low-rank" approximation. Hazan and Jaakkola (2012); Hazan et al. (2013) showed that from such an approximation, upper and lower bounds on the partition function and a sequential sampler for the Gibbs distribution can still be recovered. We show that a subfamily of our new methods, consisting of Fréchet, Exponential and Weibull tricks, can also be used with low-rank perturbations, and use these tricks to derive new upper and lower bounds on the partition function, and to construct new sequential samplers for the Gibbs distribution.

Our main contributions are as follows:

1. A family of tricks that can be implemented by simply averaging Gumbel perturbations in different spaces, and which can lead to more accurate or more sample efficient estimators of Z (Section 2.2).

2. New upper and lower bounds on the partition function of a discrete graphical model computable using low-rank perturbations, and a corresponding family of sequential samplers for the Gibbs distribution (Section 2.3).

3. Discussion of advantages of the simpler analytical form of the Gumbel trick including new links between the errors of estimating Z, sampling, and entropy estimation using low-rank Gumbel perturbations (Section 2.4).

Background and Related work The idea of perturbing the potential function of a discrete graphical model in order to sample from its associated Gibbs distribution was introduced by Papandreou and Yuille (2011), inspired by their previous work on reducing the sampling problem for Gaussian Markov random fields to the problem of finding the mean, using independent local perturbations of each Gaussian factor (Papandreou and Yuille, 2010). Tarlow et al. (2012) extended this perturb-and-MAP approach to sampling, in particular by considering more general structured prediction problems. Hazan and Jaakkola (2012) pointed out that MAP perturbations are useful not only for sampling the Gibbs distribution (considering the argmax of the perturbed model), but also for bounding and approximating the partition function (by considering the value of the max).

Afterwards, Hazan et al. (2013) derived new lower bounds on the partition function and proposed a new sampler for the Gibbs distribution that samples variables of a discrete graphical model sequentially, using expected values of low-rank MAP perturbations to construct the conditional probabilities. Due to the low-rank approximation, this algorithm has the option to reject a sample. Orabona et al. (2014) and Hazan et al. (2019) subsequently derived measure concentration results for the Gumbel distribution that can be used to control the rejection probability. Maji et al. (2014) derived an uncertainty measure from random MAP perturbations, using it within a Bayesian active learning framework for interactive image boundary annotation. The four last cited publications have been recently synthesized into a journal article (Hazan et al., 2019).

Perturb-and-MAP was famously generalized to continuous spaces by Maddison et al. (2014), replacing the Gumbel distribution with a Gumbel process and calling the resulting algorithm *A* sampling*. Maddison (2016) cast this work into a unified framework together with adaptive rejection sampling techniques, based on the notion of exponential races. This view generally brings together perturb-and-MAP and accept-reject samplers, exploiting the connection between the Gumbel distribution and competing exponential clocks that we also discuss in Section 2.2.1.

Inspired by A* sampling, Kim et al. (2016) proposed an exact sampler for discrete graphical models based on lazily-instantiated random perturbations, which uses linear programming relaxations to prune the optimization space. Further applications of perturb-and-MAP include structured prediction in computer vision (Bertasius et al., 2017) and turning the discrete sampling problem into an optimization task that can be cast as a multi-armed bandit problem (Chen and Ghahramani, 2016), see Section 2.5.2 below.

In addition to perturb-and-MAP methods, we are aware of three other approaches to estimate the partition function of a discrete graphical model via MAP solver calls, and we listed them in Section 1.4.1. These are the WISH method (weighted-integrals-and-sums-by-hashing, Ermon et al., 2013) using MAP calls on models subjected to random hash constraints, the Frank-Wolfe method that iteratively updates marginals using a constrained MAP solver and line search (Belanger et al., 2013; Krishnan et al., 2015), and finally the approach of Weller and Jebara (2014a) using a single MAP call over a discretized mesh of marginals to approximate the Bethe partition function, an approximation of the true partition function.

2.2 Relatives of the Gumbel Trick

In this section, we review the Gumbel trick and state the mechanism by which it can be generalized into an entire family of tricks. We show how these tricks can equivalently be viewed as averaging standard Gumbel perturbations in different spaces, instantiate several examples, and compare the various tricks' properties.

Notation Throughout this chapter, let $\mathcal{X}$ be a finite sample space of size $N := |\mathcal{X}|$. Let $\tilde{p} : \mathcal{X} \to [0, \infty)$ be an unnormalized mass function over $\mathcal{X}$ and let $Z := \sum_{x \in \mathcal{X}} \tilde{p}(x)$ be its normalizing partition function. Write $p(x) := \tilde{p}(x)/Z$ for the normalized version

of $\tilde{p}$, and $\phi(x) := \ln \tilde{p}(x)$ for the log-unnormalized probabilities, i.e. the potential function.

We write $\mathrm{Exp}(\lambda)$ for the exponential distribution with rate (inverse mean) λ and $\mathrm{Gumbel}(\mu)$ for the Gumbel distribution with location μ and scale 1. The latter has mean $\mu + c$, where $c := -\int_0^\infty e^{-t} \ln t \, \mathrm{d}t \approx 0.5772$ is the Euler-Mascheroni constant.

2.2.1 The Gumbel Trick

Similarly to the connection between the Gumbel trick and the Poisson process established by Maddison (2016), we introduce the Gumbel trick for discrete probability distributions using a simple and elegant construction via *competing exponential clocks*. Imagine N independent clocks, started simultaneously, such that the j-th clock rings after a random time $T_j \sim \mathrm{Exp}(\lambda_j)$. Then it is easy to show that (1) the time $\min_j\{T_j\}$ until some clock rings has $\mathrm{Exp}(\sum_{j=1}^N \lambda_j)$ distribution, and (2) the probability of the j-th clock ringing first is proportional to its rate λ_j. These properties of the exponential distribution are also widely used in survival analysis (Cox and Oakes, 1984).

Consider N competing exponential clocks $\{T_x\}_{x\in\mathcal{X}}$, indexed by elements of the domain $\mathcal{X}$, with respective rates set to the unnormalized probabilities $\lambda_x = \tilde{p}(x)$. Property (1) of competing exponential clocks tells us that

$$\min_{x\in\mathcal{X}}\{T_x\} \sim \mathrm{Exp}(Z). \tag{2.1}$$

Property (2) says that the random variable $\mathrm{argmin}_x T_x$, taking values in $\mathcal{X}$, is distributed according to p:

$$\operatorname*{argmin}_{x\in\mathcal{X}}\{T_x\} \sim p. \tag{2.2}$$

The Gumbel trick is obtained by applying the function $g(x) = -\ln x - c$ to the equalities in distribution (2.1) and (2.2). When g is applied to an $\mathrm{Exp}(\lambda)$ random variable, the result follows the $\mathrm{Gumbel}(-c + \ln\lambda)$ distribution, which can also be represented as $\ln\lambda + \gamma$, where $\gamma \sim \mathrm{Gumbel}(-c)$. Defining $\{\gamma(x)\}_{x\in\mathcal{X}} \overset{\text{iid}}{\sim} \mathrm{Gumbel}(-c)$ and noting that g is strictly decreasing, applying the function g to equalities in distribution (2.1) and (2.2), we obtain:

$$\max_{x\in\mathcal{X}}\{\phi(x) + \gamma(x)\} \sim \mathrm{Gumbel}(-c + \ln Z), \tag{2.1'}$$

$$\operatorname*{argmax}_{x\in\mathcal{X}}\{\phi(x) + \gamma(x)\} \sim p, \tag{2.2'}$$

where we have recalled that $\phi(x) = \ln \lambda_x = \ln \tilde{p}(x)$. The distribution Gumbel$(-c+\ln Z)$ has mean $\ln Z$, and thus the log partition function can be estimated by averaging samples (Hazan and Jaakkola, 2012).

2.2.2 Constructing New Tricks

Given the equality in distribution (2.1), we can treat the problem of estimating the partition function Z as a parameter estimation problem for the exponential distribution. Applying the function $g(x) = -\ln x - c$ as in the Gumbel trick to obtain a random variable distributed as Gumbel$(-c + \ln Z)$, and estimating its mean to obtain an unbiased estimator of $\ln Z$, is just one way of inferring information about Z.

We consider applying different functions g to (2.1); particularly those functions g that transform the exponential distribution to another distribution with known mean. As the original exponential distribution has rate Z, the transformed distribution will have mean $f(Z)$, where f will in general no longer be the logarithm function. Since in many scenarios we are interested in estimating various transformations $f(Z)$ of Z, this provides us a with a collection of unbiased estimators to choose from. Moreover, further transforming these estimators yields a collection of (biased) estimators for other transformations of Z, including Z itself.

Example 6 (Weibull tricks)**.** For any $\alpha > 0$, applying the function $g(x) = x^\alpha$ to an Exp(λ) random variable yields a random variable with the Weibull$(\lambda^{-\alpha}, \alpha^{-1})$ distribution with scale $\lambda^{-\alpha}$ and shape α^{-1}, which has mean $\lambda^{-\alpha}\Gamma(1+\alpha)$ and can be also represented as $\lambda^{-\alpha}W$, where $W \sim$ Weibull$(1, \alpha^{-1})$. Defining $\{W(x)\}_{x\in\mathcal{X}} \overset{\text{iid}}{\sim}$ Weibull$(1, \alpha^{-1})$ and noting that g is increasing, applying g to the equality in distribution (2.1) gives

$$\min_{x\in\mathcal{X}}\{\tilde{p}^{-\alpha}W(x)\} \sim \text{Weibull}(Z^{-\alpha}, \alpha^{-1}). \tag{2.1"}$$

Estimating the mean of a Weibull$(Z^{-\alpha}, \alpha^{-1})$ distributed random variable yields an unbiased estimator of $Z^{-\alpha}\Gamma(1+\alpha)$. The special case $\alpha = 1$ corresponds to the identity function $g(x) = x$; we call the resulting trick the *Exponential trick*. □

Table 2.1 lists several examples of tricks derived this way. As Example 6 shows, these tricks may not involve additive perturbation of the potential function $\phi(x)$; the Weibull tricks multiplicatively perturb exponentiated unnormalized probabilities $\tilde{p}^{-\alpha}$ with Weibull noise. As models of interest are often specified in terms of potential functions, to be able to reuse existing MAP solvers with the new tricks in a black-box

Table 2.1 New tricks for constructing unbiased estimators of different transformations $f(Z)$ of the partition function. The tricks are obtained by following the recipe of Section 2.2.2 and Example 6 with different choices of the function g. See Section 2.2.3 for more details on the last column.

Trick	$g(x)$	Mean $f(Z)$	Variance of $g(T)$	Asymptotic var. of $\hat{Z}$
Gumbel	$-\ln x - c$	$\ln Z$	$\frac{\pi^2}{6}$	$\frac{\pi^2}{6}Z^2$
Exponential	x	$\frac{1}{Z}$	$\frac{1}{Z^2}$	Z^2
Weibull α	x^α, $\alpha > 0$	$Z^{-\alpha}\Gamma(1+\alpha)$	$\frac{\Gamma(1+2\alpha)-\Gamma(1+\alpha)^2}{Z^{2\alpha}}$	$\frac{1}{\alpha^2}\left(\frac{\Gamma(1+2\alpha)}{\Gamma(1+\alpha)^2}-1\right)Z^2$
Fréchet α	x^α, $\alpha \in (-1,0)$	$Z^{-\alpha}\Gamma(1+\alpha)$	$\frac{\Gamma(1+2\alpha)-\Gamma(1+\alpha)^2}{Z^{2\alpha}}$ (for $\alpha > 1/2$)	$\frac{1}{\alpha^2}\left(\frac{\Gamma(1+2\alpha)}{\Gamma(1+\alpha)^2}-1\right)Z^2$
Pareto	e^x	$\frac{Z}{Z-1}$ (for $Z > 1$)	$a\frac{Z}{(Z-1)^2(Z-2)}$ (for $Z > 2$)	$\frac{Z^2}{(Z-2)^2}$
Tail t	$\mathbb{1}_{\{x>t\}}$	e^{-tZ}	$e^{-tZ}(1-e^{-tZ})$	$\frac{(1-e^{-tZ})^2}{t^2}$

manner, we seek an equivalent formulation in terms of the potential function. The following Proposition shows that by not passing the function g through the minimization in equation (2.1), the new tricks can be equivalently formulated as averaging additive Gumbel perturbations of the potential function in different spaces.

Proposition 1. *For any $g : [0,\infty) \to \mathbb{R}$ such that the expectation $f(Z) := \mathbb{E}_{T\sim\mathrm{Exp}(Z)}[g(T)]$ exists, we have*

$$f(Z) = \mathbb{E}_\gamma\left[g\left(e^{-c}\exp\left(-\max_{x\in\mathcal{X}}\{\phi(x)+\gamma(x)\}\right)\right)\right], \tag{2.3}$$

where $\{\gamma(x)\}_{x\in\mathcal{X}} \overset{iid}{\sim} \mathrm{Gumbel}(-c)$.

Proof. As $\max_x\{\phi(x)+\gamma(x)\} \sim \mathrm{Gumbel}(-c+\ln Z)$, we have

$$e^{-c}\exp\left(\max_{x\in\mathcal{X}}\{\phi(x)+\gamma(x)\}\right) \sim \mathrm{Exp}(Z) \tag{2.4}$$

and the result follows by the assumption relating f and g. □

Proposition 1 shows that the new tricks can be implemented by solving the same MAP problems $\max_x\{\phi(x)+\gamma(x)\}$ as in the Gumbel trick, and then merely passing the solutions through the function $x \mapsto g(e^{-c}\exp(x))$ before averaging them to approximate the desired expectation.

2.2.3 Comparing Tricks

Asymptotic efficiency

The Delta method (Casella and Berger, 2002) is a simple technique for assessing the asymptotic variance of estimators that are obtained by a differentiable transformation of an estimator with known variance. The last column in Table 2.1 lists asymptotic variances of corresponding tricks when unbiased estimators of $f(Z)$ are passed through the function f^{-1} to yield (biased, but consistent and non-negative) estimators of Z itself. It is interesting to examine the constants that multiply Z^2 in some of the obtained asymptotic variance expressions for the different tricks. For example, it can be shown using Gurland's ratio (Gurland, 1956) that this constant is at least 1 for the Weibull and Fréchet tricks, which is precisely the value achieved by the Exponential trick (which corresponds to $\alpha = 1$). Moreover, the Gumbel trick constant $\pi^2/6$ can be shown to be the limit as $\alpha \to 0$ of the Weibull and Fréchet trick constants. In particular, the constant of the Exponential trick is strictly better than that of the standard Gumbel trick: $1 < \pi^2/6 \approx 1.65$. This motivates us to compare the Gumbel and Exponential tricks in more detail.

Mean squared error (MSE)

For statistical estimators Y, their $\mathrm{MSE}(Y) = \mathrm{var}(Y) + \mathrm{bias}(Y)^2$ is a commonly used comparison metric. When the Gumbel or Exponential tricks are used to estimate either Z or $\ln Z$, the biases, variances, and MSEs of the estimators can be computed analytically using standard methods (see Appendix A.1 for details).

For example, the unbiased estimator of $\ln Z$ from the Gumbel trick can be turned into a consistent non-negative estimator of Z by exponentiation: $Y = \exp(\frac{1}{M}\sum_{m=1}^{M} X_m)$, where $X_1, \ldots, X_M \overset{\text{iid}}{\sim} \mathrm{Gumbel}(-c + \ln Z)$ are obtained using equation (2.1'). The bias and variance of Y can be computed using independence and the moment generating functions of the X_m's, see Appendix A.1 for the calculations.

Perhaps surprisingly, all estimator properties only depend on the true value of Z and not on the structure of the model (distribution p); this is because the estimators rely only on i.i.d. samples of a $\mathrm{Gumbel}(-c + \ln Z)$ random variable. Figure 2.1 shows the analytically computed estimator variances and MSEs for the Gumbel and Exponential tricks. For estimating Z itself (left), the Exponential trick outperforms the Gumbel trick in terms of MSE for all sample sizes $M \geq 3$ (for $M \in \{1, 2\}$, both estimators have infinite variance and MSE). The ratio of MSEs quickly approaches $\pi^2/6$, and in

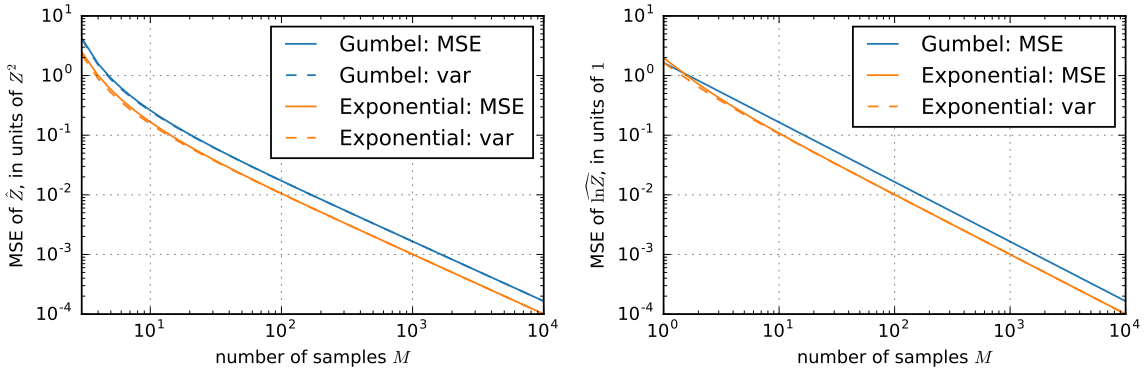


Fig. 2.1 Analytically computed MSE and variance of Gumbel and Exponential trick estimators of Z (left) and $\ln Z$ (right). The MSEs are dominated by the variance, so the dashed and solid lines mostly overlap. See Section 2.2.3 for details.

this regime the Exponential trick requires $1 - 6/\pi^2 \approx 39\%$ fewer samples than the Gumbel trick to reach the same MSE. Also, for estimating $\ln Z$, (Figure 2.1, right), the Exponential trick provides a lower MSE estimator for sample sizes $M \geq 2$; only for $M = 1$ the Gumbel trick provides a better estimator.

Note that as biases are available analytically, the estimators can be easily debiased (by subtracting their bias). One then obtain estimators with MSEs equal to the variances of the original estimators, shown dashed in Figure 2.1. The Exponential trick would then always outperform the Gumbel trick when estimating $\ln Z$, even with sample size $M = 1$.

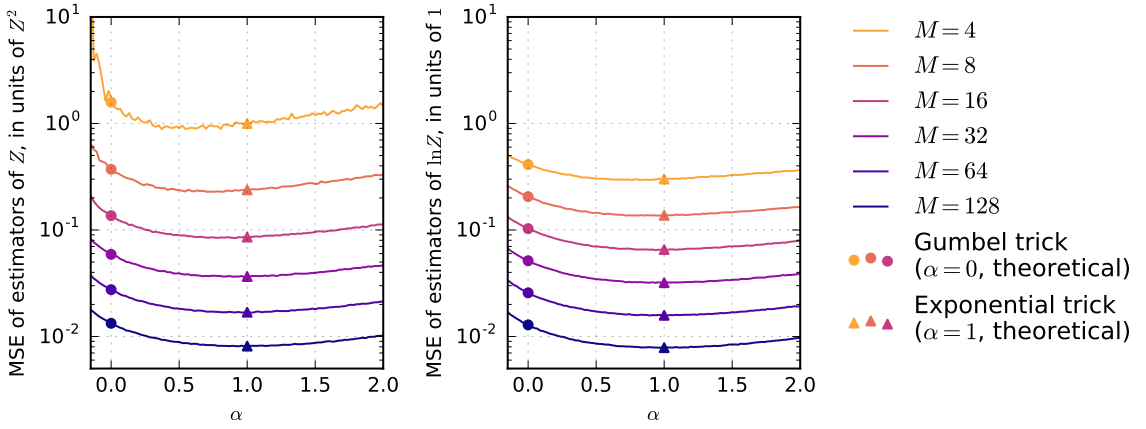


Fig. 2.2 MSE of estimators of Z (left) and $\ln Z$ (right) stemming from Fréchet ($-\frac{1}{2} < \alpha < 0$), Gumbel ($\alpha = 0$) and Weibull tricks ($\alpha > 0$). See Section 2.2.3 for details.

For Weibull tricks with $\alpha \neq 1$ and Fréchet tricks, we estimated the biases and variances of estimators of Z and $\ln Z$ by constructing $K = 100,000$ estimators in each case and evaluating their bias and variance. Figure 2.2 shows the results for varying α and several sample sizes M. We plot the analytically computed value for the Gumbel trick at $\alpha = 0$, as we observe that the Weibull trick interpolates between the Gumbel trick and the Exponential trick as α increases from 0 to 1. We note that the minimum MSE estimator is obtained by choosing a value of α that is close to 1, i.e. the Exponential trick. This agrees with the finding from Section 2.2.3 that $\alpha = 1$ is optimal as $M \to \infty$.

2.2.4 Bayesian Perspective

A Bayesian approach exposes two choices when constructing estimators of Z, or of its transformations $f(Z)$:

1. A choice of prior distribution $p_0(Z)$, encoding prior beliefs about the value of Z before any observations are collected.

2. A choice of how to summarize the posterior distribution $p_M(Z|X_1, \ldots, X_M)$ given M samples.

Taking the Jeffrey's prior $p_0(Z) \propto Z^{-1}$, an improper prior that it is invariant under reparametrization, observing M samples $X_1, \ldots, X_M \overset{\text{iid}}{\sim} \text{Exp}(Z)$ yields the posterior

$$p_M(Z|X_1, \ldots, X_M) \propto Z^{M-1} e^{-Z \sum_{m=1}^{M} X_m}. \tag{2.5}$$

Recognizing the density of a Gamma$(M, \sum_{m=1}^{M} X_m)$ random variable, the posterior mean is

$$\mathbb{E}[Z|X_1, \ldots, X_M] = \frac{M}{\sum_{m=1}^{M} X_m} = \left(\frac{1}{M} \sum_{m=1}^{M} X_m \right)^{-1}, \tag{2.6}$$

coinciding with the Exponential trick estimator of Z.

2.3 Low-rank Perturbations

One way of exploiting perturb-and-MAP to yield computational savings is to replace independent perturbations of each configuration's potential with an approximation referred to as "low-rank perturbations" in the literature (Hazan and Jaakkola, 2012).

Such approximations are available in discrete graphical models, where the domain $\mathcal{X}$ has a product space structure $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, with $\mathcal{X}_i$ the state space of the i-th variable.

Definition 1 (Hazan and Jaakkola (2012))**.** The *sum-unary perturbation MAP value* is the random variable

$$U := \max_{\mathbf{x}\in\mathcal{X}} \left\{ \phi(\mathbf{x}) + \sum_{i=1}^{n} \gamma_i(x_i) \right\}, \tag{2.7}$$

where $\{\gamma_i(x_i) \mid x_i \in \mathcal{X}_i, 1 \le i \le n\} \overset{\text{iid}}{\sim} \text{Gumbel}(-c)$.

This definition only involves $|\mathcal{X}_1| + \cdots + |\mathcal{X}_n|$ i.i.d. Gumbel random variables, rather than $|\mathcal{X}| = |\mathcal{X}_1| \times \cdots \times |\mathcal{X}_n|$. (With $n = 1$ this coincides with full-rank perturbations and $U \sim \text{Gumbel}(-c + \ln Z)$.) For $n > 2$ the distribution of U is not available analytically. One can similarly define the *pairwise* (or higher-order) *perturbations*, where independent Gumbel noise is added to each pairwise (or higher-order) potential.

Unary perturbations provide the upper bound $\ln Z \le \mathbb{E}[U]$ on the log partition function (Hazan and Jaakkola, 2012), can be used to construct a sequential sampler for the Gibbs distribution (Hazan et al., 2013), and, if the perturbations are scaled down by a factor of n, a lower bound on $\ln Z$ can also be recovered (Hazan et al., 2013). In this section we show that a subfamily of tricks introduced in Section 2.2, consisting of Fréchet and Weibull (and Exponential) tricks, is applicable in the low-rank perturbation setting and use them to derive new families of upper and lower bounds on $\ln Z$ and sequential samplers for the Gibbs distribution. Full proofs are deferred to Appendix A.2 and A.3.

2.3.1 Upper Bounds on the Partition Function

The following family of upper bounds on $\ln Z$ can be derived from the Fréchet and Weibull tricks.

Proposition 2. *For any $\alpha \in (-1, 0) \cup (0, \infty)$, the upper bound $\ln Z \le \mathcal{U}(\alpha)$ holds with*

$$\mathcal{U}(\alpha) := n\frac{\ln \Gamma(1+\alpha)}{\alpha} + nc - \frac{1}{\alpha} \ln \mathbb{E}_\gamma \left[e^{-\alpha U} \right]. \tag{2.8}$$

Proof. (Sketch.) By induction on n, with the induction step provided by our Clamping Lemma (Lemma 1) below. □

To evaluate these bounds in practice, $\mathbb{E}[e^{-\alpha U}]$ is estimated using samples of U. Corollary 9 of Hazan et al. (2019) can be used to show that $\mathrm{var}(e^{-\alpha U})$ is finite for $\alpha > -\frac{1}{2\sqrt{n}}$, and so then the estimation is well-behaved.

A natural question is how these new bounds relate to the Gumbel trick upper bound $\ln Z \leq \mathbb{E}[U]$ by Hazan and Jaakkola (2012). The following result aims to answers this.

Proposition 3. *The limit of* $\mathcal{U}(\alpha)$ *as* $\alpha \to 0$ *exists and equals* $\mathcal{U}(0) := \mathbb{E}[U]$*, i.e. the Gumbel trick upper bound.*

The question remains: when is it advantageous to use a value $\alpha \neq 0$ to obtain a tighter bound on $\ln Z$ than the Gumbel trick bound? The next result can provide guidance.

Proposition 4. *The function* $\mathcal{U}(\alpha)$ *is differentiable at* $\alpha = 0$ *and the derivative equals*

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\mathcal{U}(\alpha)\bigg|_{\alpha=0} = \frac{1}{2}\left(n\frac{\pi^2}{6} - \mathrm{var}(U)\right). \tag{2.9}$$

While the variance of U is generally not tractable, in practice one obtains samples from U to estimate the expectation in $\mathcal{U}(\alpha)$ and these samples can be reused to assess $\mathrm{var}(U)$. Interestingly, $\mathrm{var}(U)$ equals $n\pi^2/6$ for both the uniform distribution and the distribution concentrated on a single configuration, and in our empirical investigations always $\mathrm{var}(U) \leq n\pi^2/6$. In that case Proposition 4 shows that the derivative of $\mathcal{U}(\alpha)$ at 0 is non-negative and therefore Fréchet tricks ($\alpha < 0$) provide tighter bounds on $\ln Z$. However, as $\mathcal{U}(\alpha)$ is estimated with samples, the question of estimator variance arises. We investigate the trade-off between tightness of the bound $\ln Z \leq \mathcal{U}(\alpha)$ and the variance incurred in estimating $\mathcal{U}(\alpha)$ empirically in Section 2.5.3.

2.3.2 Clamping

Clamping refers to computing a quantity of interest in a graphical model with a subset of its variables fixed ("clamped") to particular values, and then aggregating the results over all possible values of the clamped variables. For example, the partition function Z of a graphical model with binary variables can be written as $Z = Z_{x_1=0} + Z_{x_1=1}$, where $Z_{x_1=v}$ is the partition function of the graphical model with variable x_1 clamped to the value v.

Consider the *partial sum-unary perturbation MAP values*, where the values of the first $j-1$ variables have been fixed to $x_1, \ldots, x_{j-1}$, and only the rest are perturbed:

$$U_j(x_1, \ldots, x_{j-1}) := \max_{x_j, \ldots, x_n} \left\{ \phi(\mathbf{x}) + \sum_{i=j}^{n} \gamma_i(x_i) \right\}. \tag{2.10}$$

The following lemma involving the U_j's serves three purposes: (I.) it provides the induction step for the proof of Proposition 2, (II.) it shows that clamping never hurts partition function estimation with Fréchet and Weibull tricks (in the sense that it cannot generate a looser upper bound), and (III.) it will be used to show that a sequential sampler constructed in Section 2.3.3 below is well-defined.

Lemma 1 (Clamping Lemma)**.** *For any $j \in \{1, \ldots, n\}$ and $(x_1, \ldots, x_{j-1}) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_{j-1}$, the following inequality holds with any $\alpha \in (-1, 0) \cup (0, \infty)$:*

$$\begin{aligned} &\sum_{x_j \in \mathcal{X}_j} \mathbb{E}_\gamma \left[e^{-(n-j)\ln\Gamma(1+\alpha) - \alpha(n-j)c)} e^{-\alpha U_{j+1}} \right]^{-1/\alpha} \\ &\leq \mathbb{E}_\gamma \left[e^{-(n-(j-1))\ln\Gamma(1+\alpha) - \alpha(n-(j-1))c)} e^{-\alpha U_j} \right]^{-1/\alpha}. \end{aligned} \tag{2.11}$$

Proof. This follows directly from the Fréchet trick ($\alpha \in (-1, 0)$) or the Weibull trick ($\alpha > 0$) and representing the Fréchet resp. Weibull random variables in terms of Gumbel random variables. See Appendix A.2.1 for more details. □

Corollary 7. *Clamping never hurts* $\ln Z$ *estimation using any of the Fréchet or Weibull upper bounds* $\mathcal{U}(\alpha)$*.*

Proof. Applying the function $x \mapsto \ln(x)$ to both sides of the Clamping Lemma 1 with $j = 1$, the right-hand side equals $\mathcal{U}(\alpha)$, while the left-hand side is the estimate of $\ln Z$ after clamping variable x_1. □

This result was shown previously in restricted settings (Hazan et al., 2013; Zhao et al., 2016). Similar results showing that clamping improves partition function estimation have been obtained for the mean field and TRW approximations (Weller and Domke, 2016), and in certain settings for the Bethe approximation (Weller and Jebara, 2014b) and L-Field (Zhao et al., 2016).

Algorithm 1 Sequential sampler for the Gibbs distribution

Input: $\alpha \in (-1, 0) \cup (0, \infty)$, potential function ϕ on $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$
Output: a sample $\mathbf{x}$ from the Gibbs distribution $\propto e^{\phi(\mathbf{x})}$

1: **for** $j = 1$ **to** n **do**
2: **for** $x_j \in \mathcal{X}_j$ **do**
3: $p_j(x_j) \leftarrow \frac{e^{-c}}{\Gamma(1+\alpha)^{1/\alpha}} \frac{\mathbb{E}_\gamma\left[e^{-\alpha U_{j+1}(x_1,\ldots,x_j)}\right]^{-1/\alpha}}{\mathbb{E}_\gamma\left[e^{-\alpha U_j(x_1,\ldots,x_{j-1})}\right]^{-1/\alpha}}$
4: $p_j(\text{reject}) \leftarrow 1 - \sum_{x_j \in \mathcal{X}_j} p_j(x_j)$
5: $x_j \leftarrow$ sample according to p_j
6: **if** $x_j ==$ reject **then**
7: RESTART (goto 1)
8: $\mathbf{x} \leftarrow (x_1, \ldots x_n)$

2.3.3 Sequential Sampling

As described in Section 1.4.2, a probability distribution $p(x_1, \ldots, x_n)$ on $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$ can be sampled using a sequential procedure where one samples X_1 from the marginal $p(x_1)$, and then for each $i = 2, \ldots, n$ samples X_i from the conditional $p(x_i \mid x_1, \ldots, x_{i-1})$. Hazan et al. (2013) derived a sequential sampling procedure for the Gibbs distribution by exploiting the $\mathcal{U}(0)$ Gumbel trick upper bound on $\ln Z$. In the same spirit, one can derive sequential sampling procedures from the Fréchet and Weibull tricks, leading to Algorithm 1.

This algorithm is well-defined if $p_j(\text{reject}) \geq 0$ for all j, which can be shown by canceling terms in the Clamping Lemma 1. We discuss correctness in Appendix A.2.2. Similarly as for the Gumbel sequential sampler of Hazan et al. (2013), the expected number of restarts (and hence the running time) only depend on the quality of the upper bound ($\mathcal{U}(\alpha) - \ln Z$), and not on the chosen ordering of variables in the graphical model.

2.3.4 Lower Bounds on the Partition Function

Similarly as in the Gumbel trick case (Hazan et al., 2013), one can derive lower bounds on $\ln Z$ by perturbing an arbitrary subset S of variables.

Proposition 5. *Let $\mathcal{X} = \mathcal{X}_1 \times \cdots \mathcal{X}_n$ be a product space and ϕ a potential function on $\mathcal{X}$. Let $\alpha \in (-1, 0) \cup (0, \infty)$. For any subset $S \subseteq \{1, \ldots, n\}$ of the variables $x_1, \ldots, x_n$ we have*

$$\ln Z \geq c + \frac{\ln \Gamma(1+\alpha)}{\alpha} - \frac{1}{\alpha} \ln \mathbb{E}\left[e^{-\alpha \max_{\mathbf{x}}\{\phi(\mathbf{x}) + \gamma_S(\mathbf{x}_S)\}}\right], \tag{2.12}$$

where $\mathbf{x}_S := \{x_i : i \in S\}$ *and* $\gamma_S(\mathbf{x}_S) \sim \text{Gumbel}(-c)$ *independently for each setting of* $\mathbf{x}_S$.

By averaging n such lower bounds corresponding to singleton sets $S = \{i\}$, we obtain a lower bound on $\ln Z$ that involves the *average-unary perturbation MAP value*

$$L := \max_{\mathbf{x} \in \mathcal{X}} \left\{ \phi(\mathbf{x}) + \frac{1}{n} \sum_{i=1}^{n} \gamma_i(x_i) \right\}. \tag{2.13}$$

Corollary 8. *For any* $\alpha \in (-1, 0) \cup (0, \infty)$*, we have the lower bound* $\ln Z \geq \mathcal{L}(\alpha)$*, where*

$$\mathcal{L}(\alpha) := c + \frac{\ln \Gamma(1+\alpha)}{\alpha} - \frac{1}{n\alpha} \ln \mathbb{E}\left[\exp\left(-n\alpha L\right)\right]. \tag{2.14}$$

Again, $\mathcal{L}(0) := \mathbb{E}[L]$ can be defined by continuity, where $\mathbb{E}[L] \leq \ln Z$ is the Gumbel trick lower bound by Hazan et al. (2013).

2.4 Advantages of the Gumbel Trick

We have seen how the Gumbel trick can be embedded into a continuous family of tricks, consisting of Fréchet, Exponential, and Weibull tricks. We showed that the new tricks can provide more efficient estimators of the partition function in the full-rank perturbation setting (Section 2.2), and in the low-rank perturbation setting lead to sequential samplers and new bounds on $\ln Z$, which can be also more efficient, as we show empirically in Section 2.5.3. To balance the discussion of merits of different tricks, in this section we briefly highlight advantages of the Gumbel trick that stem from its simpler analytical form.

First, by consulting Table 2.1 we see that the function $g(x) = -\ln x - c$ has the property that the variance of the resulting estimator (of $\ln Z$) does not depend on the value of Z; the function g is a variance stabilizing transformation for the Exponential distribution.

Second, exploiting the fact that the logarithm function leads to additive perturbations, Maji et al. (2014) showed that the entropy of x^*, the configuration with maximum potential after sum-unary perturbation in the sense of Definition 1, can be bounded as $H(x^*) \leq B(p) := \sum_{i=1}^{n} \mathbb{E}_{\gamma_i}\left[\gamma_i(x_i^*)\right]$. We extend this result to show how the errors of bounding $\ln Z$, sampling, and entropy estimation are related:

Proposition 6. *Writing p for the Gibbs distribution and $B(p) := \mathbb{E}_{\gamma_i}[\gamma_i(x_i^*)]$ for the entropy bound, we have*

$$\underbrace{(\mathcal{U}(0) - \ln Z)}_{\text{error in } \ln Z \text{ bound}} + \underbrace{\mathrm{KL}(x^* \parallel p)}_{\text{sampling error}} = \underbrace{B(p) - H(x^*)}_{\text{error in entropy estimation}} . \tag{2.15}$$

Third, the additive character of the Gumbel perturbations can also be used to derive a new result relating the error of the lower bound $\mathcal{L}(0)$ and of sampling x^{**}, defined as the configuration achieving the maximum average-unary perturbation value L, instead of exactly sampling from the Gibbs distribution p:

Proposition 7. *Writing p for the Gibbs distribution and x^{**} for the argmax in Equation (2.13),*

$$\underbrace{\ln Z - \mathcal{L}(0)}_{\text{error in } \ln Z \text{ bound}} \geq \underbrace{\mathrm{KL}(x^{**} \parallel p)}_{\text{sampling error}} \geq 0. \tag{2.16}$$

Remark. While we knew from (Hazan et al., 2013) that $\ln Z - \mathcal{L}(0) \geq 0$, this is a stronger result showing that the size of the gap is an upper bound on the KL divergence between the approximate sampling distribution of x^{**} and the Gibbs distribution p.

Proofs of both new results appear in Appendix A.2.3 and A.3.2.

Fourth, viewed as a function of the Gumbel perturbations γ, the random variable U has a bounded gradient, allowing earlier measure concentration results (Hazan et al., 2019; Orabona et al., 2014). Proving similar measure concentration results for the expectations $\mathbb{E}[e^{-\alpha U}]$ appearing in the definition of $\mathcal{U}(\alpha)$ for $\alpha \neq 0$ may be more challenging.

2.5 Experiments

We conducted experiments with the following aims:

1. To show that the higher efficiency of the Exponential trick in the full-rank perturbation setting is useful in practice, we compared it to the Gumbel trick in A* sampling (Maddison et al., 2014) (Section 2.5.1) and in the large-scale discrete sampling setting of Chen and Ghahramani (2016) (Section 2.5.2).

2. To show that non-zero values of α can lead to better estimators of $\ln Z$ in the low-rank perturbation setting as well, we compare the Fréchet and Weibull trick

bounds $\mathcal{U}(\alpha)$ to the Gumbel trick bound $\mathcal{U}(0)$ on a common discrete graphical model with different coupling strengths (Section 2.5.3).

2.5.1 A* Sampling

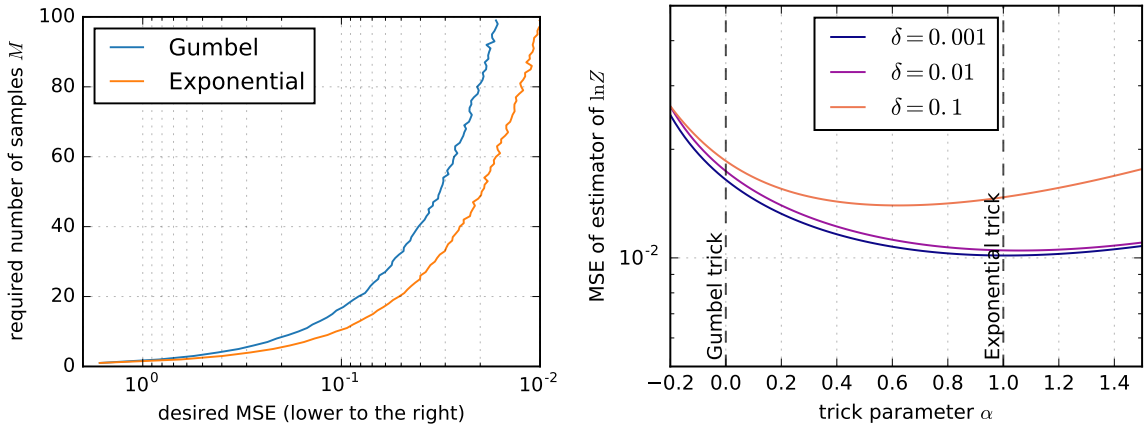
A* sampling (Maddison et al., 2014) is a sampling algorithm for continuous distributions that perturbs the log-unnormalized density ϕ with a continuous generalization of the Gumbel trick, called the Gumbel process, and uses a variant of A* search to find the location of the maximum of the perturbed ϕ. Returning the location yields an exact sample from the original distribution, as in the discrete Gumbel trick. Moreover, the corresponding maximum value also has the Gumbel$(-c + \ln Z)$ distribution (Maddison et al., 2014). Our analysis in Section 2.2.3 tells us that the Exponential trick yields an estimator with lower MSE than the Gumbel trick; we verified this on the Robust Bayesian Regression experiment of Maddison et al. (2014). We constructed estimators of $\ln Z$ from the Gumbel and Exponential tricks (debiased version, see Section 2.2.3), and assessed their variances by constructing each estimator $K = 1000$ times and looking at the sample variance. Figure 2.3a shows that the Exponential trick requires up to 40% fewer samples to reach a given MSE.

Note that despite using (an extension of) the algorithm called A* sampling, the efficiency gain arises, as in the rest of this Chapter, only when estimating normalizing constants (and not when only sampling).

2.5.2 Scalable Partition Function Estimation

Chen and Ghahramani (2016) considered sampling from a discrete distribution of the form $p(x) \propto f_0(x) \prod_{s=1}^{S} f_s(x)$ when the number of factors S is large relative to the domain size $|\mathcal{X}|$. Computing i.i.d. Gumbel perturbations $\gamma(x)$ for each $x \in \mathcal{X}$ is then relatively cheap compared to evaluating all potentials $\phi(x) = \ln f_0(x) + \sum_{s=1}^{S} \ln f_s(x)$. Chen and Ghahramani (2016) observed that each (perturbed) potential can be estimated by subsampling the factors, and potentials that appear unlikely to yield the MAP value can be pruned off from the search early on. The authors formalized the problem as a *multi-armed bandit* problem with a finite reward population and derived approximate algorithms for efficiently finding the maximum perturbed potential with a probabilistic guarantee.

While Chen and Ghahramani (2016) considered sampling, by modifying their procedure to return the value of the maximum perturbed potential rather than the



(a) (b)

Fig. 2.3 (a) Sample size M required to reach a given MSE using Gumbel and Exponential trick estimators of $\ln Z$, using samples from A^* *sampling* (see Section 2.5.1) on a Robust Bayesian Regression task. The Exponential trick is more efficient, requiring up to 40% fewer samples to reach a given MSE. (b) MSE of $\ln Z$ estimators for different values of α, using $M = 100$ samples from the approximate MAP algorithm discussed in Section 2.5.2, with different error bounds δ. For small δ, the Exponential trick is close to optimal, matching the analysis of Section 2.2.3. For larger δ, the Weibull trick interpolation between the Gumbel and Exponential tricks can provide an estimator with lower MSE.

argmax (cf equations (2.1) and (2.2)), we can estimate the partition function instead. However, the approximate algorithm only guarantees to find the MAP configuration with a probability $1 - \delta$. Figure 2.3b shows the results of running the Racing-Normal algorithm of Chen and Ghahramani (2016) on the synthetic dataset considered by the authors with the "very hard" noise setting $\sigma = 0.1$. For low error bounds δ the Exponential trick remained close to optimal, but for a larger error bound the Weibull trick interpolation between the Gumbel and Exponential tricks proved useful to provide an estimator with lower MSE.

2.5.3 Low-rank Perturbation Bounds on $\ln Z$

Hazan and Jaakkola (2012) evaluated tightness of the Gumbel trick upper bound $\mathcal{U}(0) \geq \ln Z$ on 10×10 binary spin glass models. We show one can obtain more accurate estimates of $\ln Z$ on such models by choosing $\alpha \neq 0$. To account for the fact that in practice the expectation in $\mathcal{U}(\alpha)$ is replaced with a sample average, we treat $\mathcal{U}(\alpha)$ as an estimator of $\ln Z$ with asymptotic bias equal to the bound gap $(\mathcal{U}(\alpha) - \ln Z)$, and estimate its MSE.

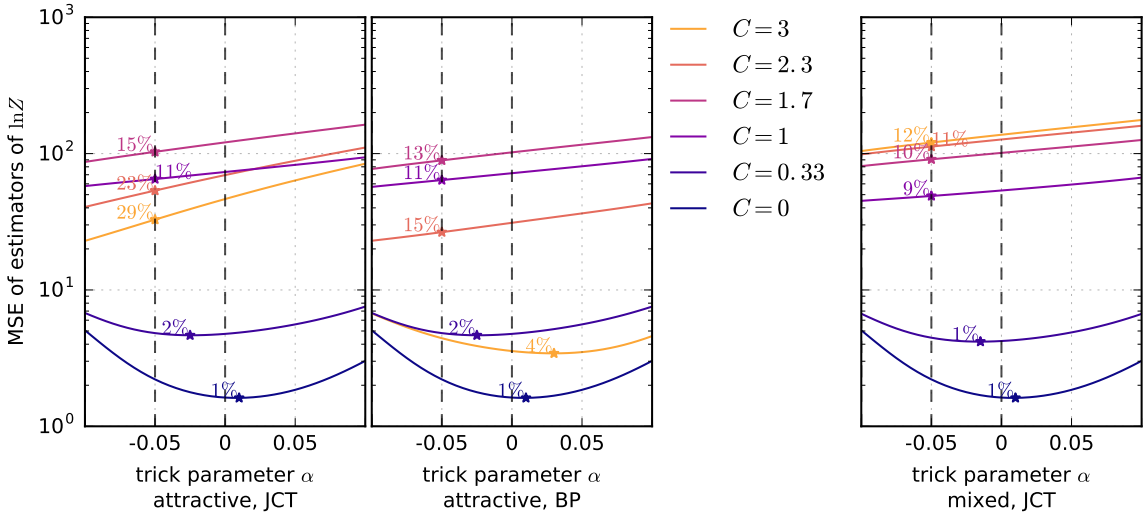


Fig. 2.4 MSEs of $\mathcal{U}(\alpha)$ as estimators of $\ln Z$ on 10×10 attractive (left, middle) and mixed (right) spin glass model with different coupling strengths C (see Section 2.5.3). We also show the percentage of samples saved by using the best α in place of the Gumbel trick estimator $\mathcal{U}(0)$, assuming the asymptotic regime. For this we only considered $\alpha > -1/(2\sqrt{n}) = -0.05$, where variance is provably finite, see Section 2.3.1. The MAP problems were solved using the exact junction tree algorithm (JCT, left and right), or approximate belief propagation (BP, middle). In all cases, when coupling is very low, α close to 0 is optimal. This also holds for BP when coupling is high. In other regimes, upper bounds derived from the Fréchet trick, i.e. $\alpha < 0$, provide more accurate estimators.

Figure 2.4 shows the MSEs of $\mathcal{U}(\alpha)$ as estimators of $\ln Z$ on 10×10 ($n = 100$) binary pairwise grid models with unary potentials sampled uniformly from $[-1, 1]$ and pairwise potentials from $[0, C]$ (*attractive* models) or from $[-C, C]$ (*mixed* models), for varying coupling strengths C. We replaced the expectations in $\mathcal{U}(\alpha)$ with sample averages of size $M = 100$, using libDAI (Mooij, 2010) to solve the MAP problems yielding these samples. We constructed each estimator 1000 times to assess its variance. The results indicate that especially in models with higher coupling strengths C, upper bounds $\mathcal{U}(\alpha)$ derived from the Fréchet trick ($\alpha < 0$) can estimate $\ln Z$ more accurately.

2.6 Discussion and Future Work

By casting partition function evaluation as a parameter estimation problem for the exponential distribution, we derived a family of methods of which the Gumbel trick is a special case. These methods can be equivalently seen as (1) perturbing models using

different distributions, or as (2) averaging standard Gumbel perturbations in different spaces, allowing implementations with essentially no additional cost.

We showed that in the full-rank perturbation setting, the new Exponential trick provides an estimator with lower MSE, or instead allows using up to 40% fewer samples than the Gumbel trick estimator to reach the same MSE.

In the low-rank perturbation setting, we used our Fréchet, Exponential and Weibull tricks to derive new bounds on $\ln Z$ and sequential samplers for the Gibbs distribution, and showed that these can also behave better than the corresponding Gumbel trick results. However, the optimal trick to use (as specified by α) depends on the model, sample size, and MAP solver used (if approximate). Since in practice the dominant computational cost is carried by solving repeated instances of the MAP problem, one can try and assess different values of α on the problem at hand. That said, we believe that investigating when different tricks yield better results is an interesting avenue for future work.

Finally, we balanced the discussion by pointing out that the Gumbel trick has a simpler analytical form which can be exploited to derive more interesting theoretical statements in the low-rank perturbation setting. Beyond existing results, we derived new connections between errors of different procedures using low-rank Gumbel perturbations.

Code to reproduce experiments presented in this chapter is available at https://github.com/matejbalog/gumbel-relatives.

Future work Several more immediate questions are left open by this work.

- *Connection to the GEV distribution*
  The Fisher–Tippett–Gnedenko theorem (Fisher and Tippett, 1928), also known as the extreme value theorem, states that the normalized maximum of i.i.d. random variables can only converge in distribution to one of three distributions: Fréchet, Gumbel, and Weibull. (As a result, these three distributions are sometimes grouped into the single Generalized Extreme Value (GEV) distribution.) These three distributions also happen to be the ones for which we were able to derive bounds on the partition function in the low-rank perturbation setting on undirected graphical models. It would be interesting to understand whether there is a common mathematical mechanism that delineates these three distributions in both of these settings.

- *Mathematical connection to α-divergences*
  If we shift our parametrization $\alpha' := \alpha + 1$ so that the Fréchet trick corresponds to $\alpha' \in (0, 1)$, the Weibull trick to $\alpha' \in (1, \infty)$, and the Gumbel trick being the limit $\alpha' \to 1$, not only is there a superficial similarity to Rényi entropies (Rényi, 1960) and α-divergences (Minka, 2005) that are defined over the same range with the Shannon entropy and Kullback-Leibler divergence the respective limits as $\alpha' \to 1$, also the mathematical expressions involved in manipulating α-divergences (e.g., Li and Gal, 2017) closely resemble those encountered by us (see Appendix A). It could be interesting to understand whether there is a deeper connection between two.

  Chapter 3 shows how the perturb-and-MAP framework can be extended to estimation of Rényi entropies. Note, however, that this extension is orthogonal to the new tricks presented in this chapter – each Rényi entropy can be estimated using any of the tricks presented here.

- *Further analysis of new tricks in the low-rank setting*
  The question when exactly an instance of the Fréchet trick leads to a more accurate estimator of the partition function has not been satisfactorily answered by this work. Further development of the analysis started by Proposition 4 could be helpful. The measure concentration results of (Orabona et al., 2014) have also not yet been extended to the new tricks.

Chapter 3

Rényi Entropy Estimation

Chapter abstract The Gumbel trick is a perturb-and-MAP method for sampling discrete distributions and estimating their normalising constants. It operates by repeatedly perturbing the potential function with Gumbel noise, and each time finding the most likely configuration. We show that a simple modification of the Gumbel trick allows unbiased estimation of Rényi entropies, and thus also obtaining bounds on the Shannon entropy. The modification can be viewed either as applying the Gumbel trick to a rescaled potential function, or equivalently as changing the scale of the applied Gumbel noise. The former view allows carrying over theoretical results about the Gumbel trick to this entropy estimation setting, while the latter view shows that from a computational perspective, our entropy estimation methods can be efficiently applied whenever the standard Gumbel trick can.

3.1 Introduction

This chapter is a continuation of Chapter 2 in that it also assumes the general setting from Section 1.5.1 where a Gibbs distribution on some sample space $\mathcal{X}$ is specified in the form of a potential function $\phi : \mathcal{X} \to [-\infty, \infty)$ that specifies the distribution via $p(\mathbf{x}) \propto e^{\phi(\mathbf{x})}$. However, this chapter is concerned with a different fundamental problem, the task of estimating the amount of uncertainty in the probability distribution p. Popular measures of uncertainty include the widely used Shannon entropy (Shannon, 1948) as well as its generalization to α-Rényi entropies (Rényi, 1960), $0 \leq \alpha \leq \infty$, where the Shannon entropy corresponds to $\alpha = 1$. As discussed in Chapter 2, *perturb-and-MAP* methods (Papandreou and Yuille, 2010) are randomized algorithms that

have already been shown useful for estimating partition functions and sampling from given distributions. The contributions of this chapter are:

1. A mathematically and computationally simple extension of the perturb-and-MAP framework to the integration problem of Rényi entropy estimation.

2. Improving the understanding of the existing Shannon entropy estimator by Maji et al. (2014), which is based on the Gumbel trick. We show that it can be recovered as the $\alpha \to 1$ limit of a specific type of our new estimators, and we also compute its variance analytically.

On a high level, this chapter shows that Rényi entropy estimation is another *integration* problem that can be converted into the *optimization* problem of finding most-likely configurations, using the perturb-and-MAP framework.

3.2 Background and Related Work

3.2.1 Rényi Entropy

The Rényi entropy (Rényi, 1960) can be seen as a generalization of the Shannon entropy into a continuous class of entropies parametrized by $\alpha \in [0, \infty]$. For $\alpha \in (0, 1) \cup (1, \infty)$ the α-Rényi entropy is given by $H_\alpha(p) := \frac{1}{1-\alpha} \ln \sum_{x \in \mathcal{X}} p(x)^\alpha$, and for $\alpha \in \{0, 1, \infty\}$ the definition is extended by taking appropriate limits, as described in subsections below.

By computing the derivative $\frac{\mathrm{d}}{\mathrm{d}\alpha} H_\alpha(p)$ one can show that Rényi entropies are non-increasing in α for $\alpha \in (0, 1)$ and $\alpha \in (1, \infty)$. Since the two-sided limit $\lim_{\alpha \to 1} H_\alpha(p)$ exists (and equals the Shannon entropy, see below) one concludes that $H_\alpha(p)$ is non-decreasing throughout $[0, \infty]$.

Rényi entropies encompass and generalize multiple measures of uncertainty with important roles in different fields, some of which we now describe.

Case $\alpha = 0$ (Hartley entropy, max-entropy)

The 0-Rényi entropy $H_0(p)$ reduces to the logarithm of the support size of p:

$$H_0(p) := \lim_{\alpha \downarrow 0} H_\alpha(p) = \ln |\operatorname{supp}(p)|. \tag{3.1}$$

This quantity is also known as the *Hartley entropy*, or the *max-entropy* of p (due to Rényi entropies being non-increasing in α).

Estimating the Hartley entropy is equivalent to *support size estimation*, an important problem well studied in the literature (Chao, 1984; Hao and Orlitsky, 2019; Valiant and Valiant, 2011; Wu and Yang, 2016). Estimating the support size is also related to the *unseen species problem* (Efron and Thisted, 1976; Orlitsky et al., 2016) in ecology, the task of estimating how many new species are going to be discovered in m future collection trials, given n existing i.i.d. observations. This problem is equivalent to support size estimation in the limit $m \to \infty$.

Case $\alpha = 1$ (Shannon entropy)

Applying l'Hôpital's rule one can check that the following two-sided limit exists:

$$H_1(p) := \lim_{\alpha \to 1} H_\alpha(p) = -\sum_{x \in \mathcal{X}} p(x) \ln p(x), \tag{3.2}$$

and equals the standard *Shannon entropy* of p. The Shannon entropy (Shannon, 1948) is a fundamental measure of uncertainty in a probability distribution with strong information-theoretic properties, and its estimation has been heavily studied in the literature (see, e.g., Paninski, 2003; Valiant and Valiant, 2011; Wu and Yang, 2016 and references therein) with applications including univariate (Vasicek, 1976) and multivariate (Goria et al., 2005) normality testing, independent component analysis (ICA, Learned-Miller and John III, 2003; Pham, 2004) as well as other signal processing applications (Hero et al., 2002), and applications in natural sciences such as studying heart rate patterns in medicine (Porta et al., 2001).

Case $\alpha = 2$ (Rényi's quadratic entropy, Collision entropy)

Known as *Rényi's quadratic entropy* due to its mathematical form, it is also called the *collision entropy* of p thanks to its intuitive interpretation as (the negative logarithm of) the probability that two independent draws from p take on the same value:

$$H_2(p) = -\ln \sum_{x \in \mathcal{X}} p(x)^2 = -\ln \mathbb{P}_{Y,Z \overset{\text{iid}}{\sim} p}[Y = Z]. \tag{3.3}$$

This is directly related to the *inverse participation ratio* $\text{IPR} := 1/(\sum_{x \in \mathcal{X}} p(x)^2) = e^{H_2(p)}$, which is used in quantum mechanics to quantify the uncertainty about the location of a particle that can exist in a quantum superposition of states over many locations.

The collision probability itself is called the *index of coincidence* $\mathrm{IC} := \sum_{x\in\mathcal{X}} p(x)^2$ and is used in natural language analysis, as well as in cryptography for the analysis of encrypted texts (Friedman, 1922). The index of coincidence can be directly computed from the collision entropy via $\mathrm{IC} = e^{-H_2(p)}$.

In the continuous case, Rényi's (differential) quadratic entropy can be conveniently estimated using a Parzen window estimator (kernel density estimator) of the distribution with a Gaussian kernel (Principe and Xu, 1999); this is thanks to the integral of a product of two Gaussian densities being analytically tractable.

Case $\alpha = \infty$ (Min-entropy)

Sandwiching the summation appearing in the Rényi entropy using the simple inequalities $(\max_{y\in\mathcal{X}} p(y))^\alpha \leq \sum_{x\in\mathcal{X}} p(x)^\alpha \leq |\mathcal{X}|(\max_{y\in\mathcal{X}} p(y))^\alpha$, one can demonstrate the limit

$$H_\infty(p) := \lim_{\alpha\to\infty} H_\alpha(p) = -\ln \max_{x\in\mathcal{X}} p(x), \tag{3.4}$$

and its value is also called the *min-entropy*, as it is the smallest among all Rényi entropies. The min-entropy plays an important role in the study of randomness extractors (Nisan, 1996; Santha and Vazirani, 1986), algorithms that turn a weakly random (biased) source into an (almost) uniformly distributed random stream, as efficiently as possible.

General Rényi entropies

Apart from being used to bound and estimate the more widely used Shannon entropy (e.g., Harvey et al. (2008) considered $\alpha \in (0,1) \cup (1,2)$ for this purpose), general α-Rényi entropies have also turned out to be useful in and of themselves.

Hero and Michel (1999) discovered a relationship between the Rényi entropy and the edge weight of approximate minimum trees spanning a set of samples from a given probability distribution over Euclidean space. Ma et al. (2000) utilized this relationship for *image registration*, the task of aligning multiple images taken from different angles at different times, by minimizing the Rényi entropy of the joint distribution of aligned images. The general approach of *entropic spanning graphs* estimating Rényi entropies has subsequently been utilized in many other general tasks including feature clustering and image retrieval (Hero et al., 2002).

General Rényi entropies have also found applications in information theory (Csiszár, 1995), quantum physics (Cui et al., 2012), and unsupervised clustering (Jenssen et al.,

2003). In both classical and quantum thermodynamics, the α-Rényi entropy can be interpreted as the maximum amount of work a system whose temperature is suddenly divided by α is able to perform as it moves into its new equilibrium, divided by the absolute change in temperature (Baez, 2011). The asymptotic behaviour of the Rényi entropy of a specific Ising model was studied analytically by Stéphan et al. (2010).

3.2.2 Estimating Rényi Entropies

Apart from estimation methods specific to particular values of α already mentioned in Section 3.2.1, the problem of estimating Rényi entropies for general values of α has also been studied in the literature.

Similarly as for the Shannon entropy, literature on Rényi entropy estimation usually assumes access to N samples $X_1, \ldots, X_N \overset{\text{iid}}{\sim} p$ from the unknown distribution p, and the task is to estimate the Rényi entropy from these N points only. This setup includes the aforementioned work on *entropic spanning graphs* (Hero et al., 2002), as well as later work on estimators based on k-nearest neighbour graphs for which Pál et al. (2010) proved consistency of the estimators and also provided first finite-sample error bounds. Acharya et al. (2016) identified three different regimes in which sample complexity of Rényi entropy estimation behaves surprisingly differently: $\alpha < 1$, non-integer $\alpha > 1$ and integer $\alpha > 1$. Some of their sample complexity bounds were subsequently refined by Obremski and Skorski (2017). The most recent upper bounds are due to Hao and Orlitsky (2019), who showed that they can be achieved by *profile maximum likelihood* (Orlitsky et al., 2004), a plug-in estimation method where a distribution maximizing the likelihood of the observed sample *profile* ("histogram of histograms") is found. A major advantage of this approach is that once a profile maximum likelihood estimate is computed, it can be used to estimate multiple properties of the unknown distribution at once, including Rényi entropies of different orders α (Hao and Orlitsky, 2019).

A somewhat different setup is considered in the *streaming setting*, where one still assumes i.i.d. samples from p but these arrive in an online manner and there are time and memory constraints imposed on the computation carried out after each sample arrives. The problem is related to estimating frequency moments of a data stream (Alon et al., 1999; Indyk and Woodruff, 2005): given an incoming sequence with n_x elements equal to x, the task is to estimate moments of the form $F^\alpha := \sum_x n_x^\alpha$. Harvey et al. (2008) exploited this relationship for $\alpha \in (0, 1) \cup (1, 2]$ and showed how the corresponding Rényi entropies can be used to estimate the Shannon entropy in the streaming setting;

however, the subroutine estimating the Rényi entropy is also independently applicable. More recently, among other distribution properties Crouch et al. (2016) considered estimating from a stream the collision probability of a distribution, which is directly related to the Rényi entropy with $\alpha = 2$ (see Section 3.2.1).

In this work we consider a substantially different setup, where instead of access to samples from p we assume that the distribution is described by a potential function $\phi : \mathcal{X} \to [-\infty, \infty)$. This setup arises in the general framework of graphical models (Section 1.5.1), and is identical to the setup of Chapter 2. Similarly to partition function estimation, computing entropies in this setting is a high-dimensional integration problem that requires consideration of all possible configurations of the graphical model in the worst case. For the Shannon entropy, a popular approximation is the Bethe entropy, which only sums up singleton entropies of univariate marginals and mutual informations between pairs of variables joined by edges of the graphical model. The closest work to ours is that of Maji et al. (2014), who exploited the duality relationship between the partition function and the Shannon entropy to construct a perturb-and-MAP estimator of the Shannon entropy based on the Gumbel trick; we discuss its relationship to our Rényi entropy estimators in detail in Section 3.4.1.

3.3 Method

Notation Throughout this chapter, we assume the following setup. Let $\mathcal{X}$ be a finite set, $\tilde{p} : \mathcal{X} \to [0, \infty)$ an unnormalized mass function over $\mathcal{X}$, $\phi : \mathcal{X} \to [-\infty, \infty)$ the associated potential function $\phi(\mathbf{x}) := \ln \tilde{p}(\mathbf{x})$, and $p = \tilde{p}/Z$ the normalized version of $\tilde{p}$. The task is to estimate the α-Rényi entropy of p, for a given value of α.

Main idea As a first step, assume that $\tilde{p} = p$ is in fact normalized, so $Z = 1$ and $\phi(\mathbf{x}) = \ln p(\mathbf{x})$. In that case the α-Rényi entropy of p, where $\alpha \in [0, 1) \cup (1, \infty)$, can be rewritten using the Gumbel trick applied to the unnormalized mass function

$\mathbf{x} \mapsto p(\mathbf{x})^\alpha$:

$$H_\alpha(p) := \frac{1}{1-\alpha} \ln \sum_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x})^\alpha \tag{3.5}$$

$$= \frac{1}{1-\alpha} \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \ln \left(p(\mathbf{x})^\alpha \right) + \gamma(\mathbf{x}) \right\} \right] \tag{3.6}$$

$$= \frac{1}{1-\alpha} \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \alpha\phi(\mathbf{x}) + \gamma(\mathbf{x}) \right\} \right] \tag{3.7}$$

$$= \frac{\alpha}{1-\alpha} \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \phi(\mathbf{x}) + \frac{1}{\alpha}\gamma(\mathbf{x}) \right\} \right], \tag{3.8}$$

where $\{\gamma(\mathbf{x})\}_{\mathbf{x}\in\mathcal{X}} \overset{\text{iid}}{\sim} \text{Gumbel}(-c)$ are i.i.d. Gumbel perturbations with zero mean. □

This shows that Rényi entropy estimation can be reduced to the MAP problem much in the same way as partition function estimation was by Hazan and Jaakkola (2012). While Equation (3.7) shows that the Rényi entropy can be estimated by applying the Gumbel trick to a rescaled potential function $\alpha\phi(\mathbf{x})$, Equation (3.8) shows that in fact one can keep working with the unmodified potential function $\phi(\mathbf{x})$ and instead just change the scaling of the Gumbel perturbations. The former view is important as it allows carrying over theoretical results about the Gumbel trick to the Rényi entropy estimation setting (the trick is applied unchanged, only to a rescaled potential function). The latter view is useful from an implementation perspective, as it shows that the method can be easily applied whenever the original Gumbel trick can; the only required modification is scaling the Gumbel perturbation by $\frac{1}{\alpha}$.

Note that estimating the Rényi entropy by sampling from p and estimating the α-moment of p from samples would not lead to an unbiased estimator of the Rényi entropy $H_\alpha(p)$ due to the presence of the logarithm function.

Finally, as pointed out in Chapter 2, a lower-variance (but biased) estimator could be obtained by using the so-called "Exponential trick" instead of the Gumbel trick in deriving Equation (3.6).

3.3.1 The Unnormalized Case

While the above derivation for the normalized case $\tilde{p} = p$ captures the simplicity of extending the Gumbel trick to Rényi entropy estimation, in practice one is usually given an unnormalized potential function – especially in the case of undirected graphical models (Section 1.5.1). In this subsection we thus explore the case where the normalization Z is unknown. While in principle any estimation method for Z could be

used to first normalize the distribution, we show that also using the Gumbel trick for this purpose is conveniently possible, and statistically advantageous.

In the unnormalized case the α-Rényi entropy $H_\alpha(p)$ can be decomposed into two terms, the first involving the unnormalized distribution $\tilde{p}$ and the second involving the unknown partition function Z. The first term is then handled mathematically in the same way as in the normalized case above, while the log-partition function $\ln Z$ in the second term is re-expressed using the standard Gumbel trick:

$$H_\alpha(p) := \frac{1}{1-\alpha} \ln \sum_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x})^\alpha \tag{3.9}$$

$$= \frac{1}{1-\alpha} \ln \sum_{\mathbf{x}\in\mathcal{X}} \tilde{p}(\mathbf{x})^\alpha - \frac{\alpha}{1-\alpha} \ln Z \tag{3.10}$$

$$= \frac{\alpha}{1-\alpha} \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \phi(\mathbf{x}) + \frac{1}{\alpha}\gamma(\mathbf{x}) \right\} \right] - \frac{\alpha}{1-\alpha} \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \{\phi(\mathbf{x}) + \gamma(\mathbf{x})\} \right]. \tag{3.11}$$

As in partition function estimation (Chapter 2), the expectations will utimately be estimated using $M \in \mathbb{N}$ samples. Here, the two expectations could be estimated separately, with independent Gumbel perturbations in both cases. However, by *linearity of expectation* the difference of expectations will be the same if Gumbel perturbations $\gamma(\mathbf{x})$ are shared between the two terms:

$$H_\alpha(p) = \frac{\alpha}{1-\alpha} \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \phi(\mathbf{x}) + \frac{1}{\alpha}\gamma(\mathbf{x}) \right\} - \max_{\mathbf{x}\in\mathcal{X}} \{\phi(\mathbf{x}) + \gamma(\mathbf{x})\} \right]. \tag{3.12}$$

The second term, which accounts for the unknown normalisation Z, can intuitively be seen as a control variate since it appears to be positively correlated with the first term. The following proposition formalizes this intuition and shows that indeed sharing the noise cannot increase the estimator variance:

Proposition 8. *For $A \subseteq \mathcal{X}$, let $\hat{H}_\alpha^A(p)$ be the estimator of $H_\alpha(p)$ that shares the noise among configurations in A, and uses independent noise for the others:*

$$\hat{H}_\alpha^A(p) := \frac{\alpha}{1-\alpha} \frac{1}{M} \sum_{m=1}^{M} \left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \phi(\mathbf{x}) + \frac{1}{\alpha}\gamma_0^{(m)}(\mathbf{x}) \right\} - \max_{\mathbf{x}\in\mathcal{X}} \left\{ \phi(\mathbf{x}) + \gamma_1^{(m)}(\mathbf{x}) \right\} \right] \tag{3.13}$$

where $\gamma_0^{(m)}(\mathbf{x}) = \gamma_1^{(m)}(\mathbf{x})$ for all $\mathbf{x} \in A$, and otherwise all γ's are i.i.d. Gumbel$(-c)$. *Then*

$$\operatorname{var}\left[\hat{H}_\alpha^{\mathcal{X}}(p)\right] \leq \operatorname{var}\left[\hat{H}_\alpha^{\emptyset}(p)\right] = \frac{1}{M} \frac{1+\alpha^2}{(1-\alpha)^2} \frac{\pi^2}{6} \tag{3.14}$$

Proof (sketch). The product measure version of Harris' inequality (a special case of the FKG inequality) can be used to show that the covariance between the two terms is non-negative; see Appendix B for details. □

While the control variate appeared out of necessity to estimate the unknown log-partition function $\ln Z$, it can also be beneficial when Z is already known. Analytical treatment of the general case is difficult, but it can be shown that a lower-variance estimator is obtained when $\alpha < 2$ and the distribution is uniform on its support (which includes the extreme cases of deterministic and uniform distributions as special cases). See Proposition 19 in Appendix B for details.

Another potential advantage stemming from sharing Gumbel perturbations between the two estimation problems is that depending on the procedure used, the two maximization problems could be solved jointly since especially for values of α close to 1 the optimization landscapes of the two problems are similar.

Limiting cases

For $\alpha \to 0$ (max-entropy) our Rényi entropy estimator eliminates the potential function from the entropy estimation term (multiplies it by 0, see Equation (3.7)) and does not require the control variate term at all ($\frac{\alpha}{1-\alpha} = 0$). The method is thus prescribing uniform sampling from the distribution's support, and looking at the value of the largest perturbation. The variance of the resulting estimator is $\frac{\pi^2}{6}$ by max-stability of the Gumbel distribution. This highlights a difference to entropy estimation from samples, where estimating the max-entropy is a potentially challenging support size estimation problem (see Section 3.2.1). The reason behind this difference is that in the perturb-and-MAP setting we are able to change the sampling distribution.

For $\alpha \to \infty$ (min-entropy) the fraction $\frac{\alpha}{1-\alpha} \to 1$, and the multiplicative factor of the Gumbel perturbation in the entropy estimation term vanishes $\frac{1}{\alpha} \to 0$. The method thus prescribes turning off the Gumbel perturbation and simply finding the configuration with maximum potential. When the normalization Z is unknown, the Gumbel trick is only needed to estimate $\ln Z$; otherwise the estimator of $H_\infty(p)$ is fully deterministic (has variance 0). Again this highlights the difference from entropy estimation using samples, since in the Perturb-and-MAP setting tractable identification of maximum-probability configurations is assumed.

Finally, the limit $\alpha \to 1$ (Shannon entropy) will be studied in more detail in Section 3.4.

3.3.2 Gumbel Trick Machinery in Rényi Entropy Estimation

In this subsection we discuss existing theoretical and algorithmic developments related to the Gumbel trick, and show that they cleanly extend to the new Rényi entropy estimation setting considered in this chapter.

Relatives of the Gumbel Trick

Chapter 2 generalized the Gumbel trick into an entire family of tricks, termed the "relatives" of the Gumbel trick. Other members of this family can be useful for two reasons:

1. they provide *unbiased* estimators of transformations of Z other than $\ln Z$, and
2. new estimators, even if biased, can have lower variance than the Gumbel trick estimator for the same quantity.

The Gumbel trick estimators generally lead to unbiased estimators of the Rényi entropy, but as an example of (1), the *Exponential trick* allows unbiased estimation of the *inverse participation ratio* $e^{H_2(p)} = (\sum_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x})^2)^{-1}$, which is of interest in quantum mechanics.

As an example of (2), the *Exponential trick* provides a biased estimator of the Rényi entropy but its MSE will be provably lower than the MSE (variance) of the Gumbel trick estimator presented above (this follows from Section 2.2.3). In the low-rank perturbation setting in discrete graphical models, it is the *Fréchet trick* that tends to yield estimators with lowest MSE (see Section 2.5.3 and Figure 2.3).

Low-rank Setting

Equation (3.8) showed that the Rényi entropy can be obtained by simply applying the standard Gumbel trick to a rescaled potential function $\alpha\phi(\mathbf{x})$. As a result, the low-rank perturbation upper and lower bounds on the log-partition function derived either from the Gumbel trick (Hazan and Jaakkola, 2012; Hazan et al., 2013) or from its relatives (Chapter 2) can be directly applied to yield bounds on the Rényi entropy $H_\alpha(p)$. For example, from the Gumbel trick low-rank bounds we have (for simplicity,

in the normalized case $\phi(\mathbf{x}) = \ln p(\mathbf{x})$):

$$\begin{aligned}&\frac{\alpha}{1-\alpha}\mathbb{E}_\gamma\left[\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \frac{1}{\alpha n}\sum_{i=1}^{n}\gamma_i(x_i)\right\}\right]\\ &\leq H_\alpha(p) \leq \frac{\alpha}{1-\alpha}\mathbb{E}_\gamma\left[\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \frac{1}{\alpha}\sum_{i=1}^{n}\gamma_i(x_i)\right\}\right]\end{aligned} \tag{3.15}$$

when $\alpha \in [0, 1)$, while for $\alpha \in (1, \infty)$ these inequalities hold reversed (since $\frac{\alpha}{1-\alpha} < 0$). In the unnormalized case where Z is not known the standard low-rank bounds on $\ln Z$ can be used, and Gumbel perturbations shared between the two expectations as in Section 3.3.1.

Being able to bound Rényi entropies is particularly useful if the Shannon entropy is of interest, as for $\alpha_1 < 1$ upper bounds on $H_{\alpha_1}(p)$ yield upper bounds on the Shannon entropy, while for $\alpha_2 > 1$ lower bounds on $H_{\alpha_2}(p)$ yield lower bounds on the Shannon entropy:

$$\begin{aligned}&\frac{\alpha_2}{1-\alpha_2}\mathbb{E}_\gamma\left[\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \frac{1}{\alpha_2}\sum_{i=1}^{n}\gamma_i(x_i)\right\}\right]\\ &\leq H_{\alpha_2}(p) \leq H_1(p) \leq H_{\alpha_1}(p) \leq \frac{\alpha_1}{1-\alpha_1}\mathbb{E}_\gamma\left[\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \frac{1}{\alpha_1}\sum_{i=1}^{n}\gamma_i(x_i)\right\}\right].\end{aligned} \tag{3.16}$$

(Again the extension to the unnormalized case is handled by adding the corresponding low-rank bounds on $\ln Z$.) This situation is different from (Maji et al., 2014) where a low-rank estimator of the Shannon entropy is also constructed, but it is only an upper bound on the Shannon entropy of an approximate distribution, and not on the Shannon entropy of the true distribution p.

Measure concentration

In practice the expectations with respect to Gumbel perturbations need to be estimated by samples. Since the Gumbel distribution has a heavy upper tail, the question of measure concentration arises. Orabona et al. (2014) and Hazan et al. (2019) proved that samples of max-perturbations do indeed concentrate in the low-rank setting. Thanks to the perspective of applying the standard Gumbel trick to a rescaled potential function, these results directly translate to the low-rank bounds on Rényi entropies from the preceding section.

Continuos case

Maddison et al. (2014) generalized Gumbel trick sampling to continuous distributions: the log-unnormalized density ϕ is additively perturbed by a continuous generalization γ of the Gumbel trick, called the *Gumbel process*, and a variant of A* search is used to find the location of the maximum of the perturbed $\phi + \gamma$. This location is an exact sample from the original Gibbs distribution $p \propto e^{\phi}$, as in the discrete Gumbel trick. Analogously, the corresponding maximum value follows the Gumbel$(-c + \ln Z)$ distribution (Maddison et al., 2014), where Z is the normalized of the unnormalized density e^{ϕ}.

The mathematics of Equations (3.6-3.8) applies unchanged when the summation over a discrete space is replaced by integration over a continuous $\mathcal{X}$. The quantity being thus estimated is sometimes called the *differential Rényi entropy*. As in the discrete setting, apart from its own uses it can also be used to estimate the (differential) Shannon entropy.

3.4 Shannon Entropy Estimation

Since the Rényi entropy $H_\alpha(p)$ of a fixed distribution p is non-decreasing in α, and $\alpha = 1$ corresponds to the Shannon entropy $H_1(p)$, one can estimate the latter both from below and from above using Rényi entropies: $H_{\alpha_2}(p) \leq H_1(p) \leq H_{\alpha_1}(p)$ for all $\alpha_1 \in [0, 1)$, $\alpha_2 \in (1, \infty]$. The Gumbel trick approaches from Section 3.3 can be used to estimate $H_{\alpha_1}(p)$ and $H_{\alpha_2}(p)$, and as $\alpha_1 \uparrow 1$ and $\alpha_2 \downarrow 1$ the error due to the Rényi approximation vanishes. However, when using the normalized case estimators there is a trade-off: as $\alpha \to 1$, the estimator variance diverges to ∞. Perhaps surprisingly, the control variate from Section 3.3.1 can be used to stabilize the estimator as $\alpha \to 1$. In fact, we shall see that in the limit $\alpha \to 1$ our Rényi entropy estimators recover the Gumbel trick estimator of the Shannon entropy due to Maji et al. (2014).

3.4.1 Relationship to the Maji et al. (2014) Estimator

Maji et al. (2014) have shown that the value of the perturbation at the argmax location $\mathbf{x}^*$ can be used as an unbiased estimator $\hat{H}_1 := \gamma(\mathbf{x}^*)$ of the Shannon entropy:

$$\mathbb{E}\left[\hat{H}_1\right] = \mathbb{E}\left[\gamma(\mathbf{x}^*)\right] = H_1(p) \qquad \text{where} \qquad \mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}\in\mathcal{X}} \left\{\phi(\mathbf{x}) + \gamma(\mathbf{x})\right\}. \tag{3.17}$$

In this section we set out to answer the natural question of how our Rényi entropy estimators $\hat{H}_\alpha$ (with $\alpha \in (0,1) \cup (1,\infty)$) relate to the Shannon entropy estimator $\hat{H}_1$.

Proposition 9. *As $\alpha \to 1$, our variance-reduced Rényi entropy estimator $\hat{H}_\alpha$ converges almost surely to Maji's Shannon entropy estimator $\hat{H}_1$ based on the same Gumbel perturbations.*

Proof. By linearity, it suffices to show the result for the estimators based on $M = 1$ samples. Let $\Omega_1 \subseteq \Omega$ be a subset of the probability space with $\mathbb{P}(\Omega_1) = 1$ such that for all $\omega \in \Omega_1$, the argmax defining $\mathbf{x}^*$ in Equation (3.17) is unique. On this subset of the probability space, for all α close enough to 1, we have that

$$\operatorname*{argmax}_{\mathbf{x}\in\mathcal{X}} \left\{\phi(\mathbf{x}) + \frac{1}{\alpha}\gamma(\mathbf{x})\right\} = \operatorname*{argmax}_{\mathbf{x}\in\mathcal{X}} \{\phi(\mathbf{x}) + \gamma(\mathbf{x})\} =: \mathbf{x}^*, \tag{3.18}$$

and therefore for these values of α,

$$\begin{aligned}
\hat{H}_\alpha(p) &:= \frac{\alpha}{1-\alpha}\left[\max_{\mathbf{x}\in\mathcal{X}} \left\{\phi(\mathbf{x}) + \frac{1}{\alpha}\gamma(\mathbf{x})\right\} - \max_{\mathbf{x}\in\mathcal{X}} \{\phi(\mathbf{x}) + \gamma(\mathbf{x})\}\right] && (3.19)\\
&= \frac{\alpha}{1-\alpha}\left[\left\{\phi(\mathbf{x}^*) + \frac{1}{\alpha}\gamma(\mathbf{x}^*)\right\} - \{\phi(\mathbf{x}^*) + \gamma(\mathbf{x}^*)\}\right] && (3.20)\\
&= \frac{\alpha}{1-\alpha}\frac{1-\alpha}{\alpha}\gamma(\mathbf{x}^*) = \gamma(\mathbf{x}^*) = \hat{H}_1(p), && (3.21)
\end{aligned}$$

and so the claim follows. □

This implies that the stochastic process $\{\hat{H}_\alpha \mid \alpha \geq 0\}$ has continuous sample paths almost surely. Its boundary values are marginally distributed as $\hat{H}_0 \sim \text{Gumbel}(-c + \ln N)$ and $\hat{H}_\infty \sim \text{Gumbel}(-c - \ln \max_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x}))$, but intermediate values are not necessarily marginally Gumbel. Figure 3.1 shows $M = 30$ sample paths on a synthetic, truncated geometric distribution p.

3.4.2 Variance of the Maji et al. (2014) Estimator

In this section we provide a new result about the $\hat{H}_1$ estimator, namely its variance. Recall from (Maddison, 2016) and Chapter 2 that the Gumbel trick theory can be developed using the classical notion of competing exponential clocks. In particular, the sampling result follows from the fact that the first clock to ring is distributed proportionally to the rates of the individual clocks, and the partition function estimation result follows from the fact that the time of the first ring has $\text{Exp}(Z)$ distribution,

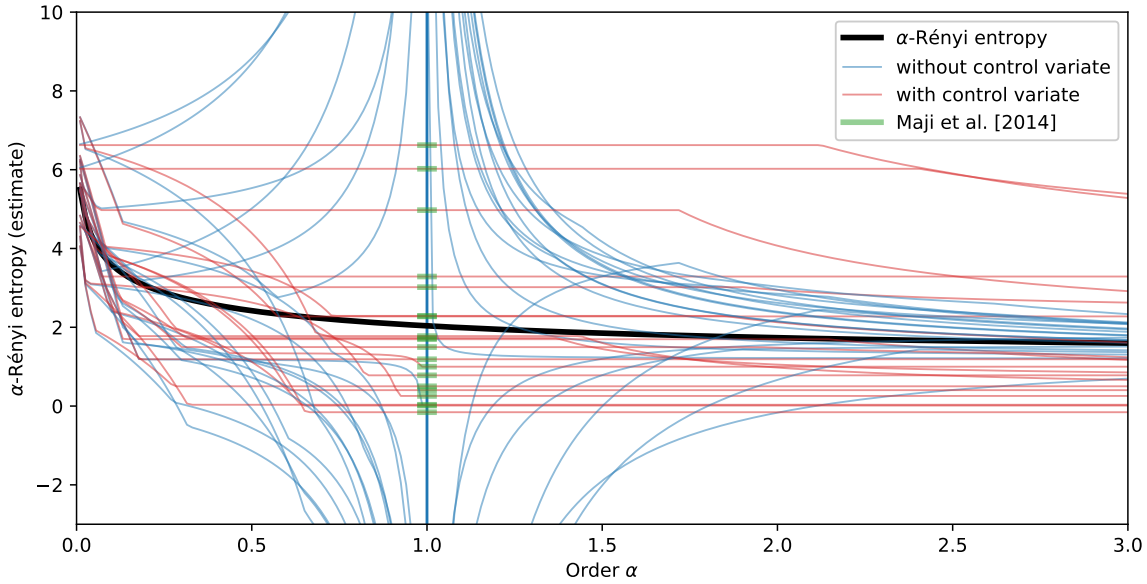


Fig. 3.1 Estimates of α-Rényi entropy for varying values of α, computed on a synthetic geometric distribution with success probability $q = 0.3$, truncated to $\{1, 2, \ldots, 500\}$ and normalized. The exact value of the entropies is plotted in thick black. $M = 30$ sets of random perturbations have been sampled. For each set, the (Maji et al., 2014) estimate of the Shannon entropy is plotted at $\alpha = 1$ in green, and the sample paths across varying α of the normalized case Rényi entropy estimator (without the control variate, shown in blue) and with the control variate (Section 3.3.1, shown in red) are plotted. As predicted by Proposition 9 the latter sample paths pass through the (Maji et al., 2014) estimates at $\alpha = 1$.

where Z is the sum of the clock rates. However, it is additionally true that the identity of the first clock to ring and the first ringing time are independent random variables. This additional property allows us to compute the variance of the Shannon entropy estimator $\hat{H}_1$.

Theorem 9. *Let p be a discrete probability distribution. The variance of the Shannon entropy estimator $\hat{H}_1$ based on (Maji et al., 2014) is*

$$\operatorname{var}\left[\hat{H}_1\right] = \frac{\pi^2}{6} + \operatorname{var}_{X\sim p}\left[\ln p(X)\right]. \tag{3.22}$$

Proof. By Eve's law (Law of Total Variance) we have after conditioning on $\mathbf{x}^*$ that

$$\operatorname{var}\left[\hat{H}_1\right] = \operatorname{var}\left[\gamma(\mathbf{x}^*)\right] = \mathbb{E}\left[\operatorname{var}\left[\gamma(\mathbf{x}^*) \mid \mathbf{x}^*\right]\right] + \operatorname{var}\left[\mathbb{E}\left[\gamma(\mathbf{x}^*) \mid \mathbf{x}^*\right]\right]. \tag{3.23}$$

By independence, given $\mathbf{x}^*$, the value of the maximum is still distributed according to $\phi(\mathbf{x}^*) + \gamma(\mathbf{x}^*) \sim \text{Gumbel}(-c - \ln Z)$ and therefore $\gamma(\mathbf{x}^*)$ has conditional expectation

$\ln Z - \phi(\mathbf{x}^*)$ and conditional variance $\frac{\pi^2}{6}$. Plugging into Equation (3.23),

$$\operatorname{var}\left[\hat{H}_1\right] = \mathbb{E}\left[\frac{\pi^2}{6}\right] + \operatorname{var}_{\mathbf{x}^*}\left[\ln Z - \phi(\mathbf{x}^*)\right] = \frac{\pi^2}{6} + \operatorname{var}_{X\sim p}\left[\ln p(X)\right]. \tag{3.24}$$

□

Remark. Note that the estimator $\hat{H}_1$ is directly applicable if the normalisation of p is unknown. The right-hand side of Equation (3.22) thus admits the following interpretation:

1. The first term $\frac{\pi^2}{6}$ is the cost of not knowing the normalisation of p; it equals the variance of the Gumbel trick estimator of $\ln Z$.

2. The second term $\operatorname{var}_{X\sim p}\left[\ln p(X)\right]$ is the variance of the following estimator $\hat{H}$ for the Shannon entropy of a *normalised* distribution: sample $X \sim p$ and set $\hat{H} := -\ln p(X)$. (This is unbiased by definition of Shannon entropy.) □

3.5 Discussion and Future Work

We showed how the Gumbel trick can be cleanly extended from partition function estimation to Rényi entropy estimation. Thanks to the simple form of the extension, most developments from the Gumbel trick literature carry over to the Rényi entropy estimation setting. These include stochastic bounds in low-rank approximations for discrete graphical models (Hazan and Jaakkola, 2012; Hazan et al., 2013), measure concentration results for these approximations (Orabona et al., 2014), extensions to the continuous case (based on A^* sampling, Maddison et al., 2014) and to other relatives of the Gumbel trick (Chapter 2). We studied the statistical properties of the derived estimators of the Rényi entropy, and characterized their relationship to the Gumbel trick-based estimator of the Shannon entropy due to Maji et al. (2014). Along the way we have also added to the understanding of properties of this Shannon entropy estimator.

Future work This chapter has focused on tackling a specific class of integration problems (Rényi entropy estimation) by generalizing a specific class of partition function estimation methods (the Gumbel trick and its "relatives") in order to obtain optimization problems with more appealing properties (at least in the specific situations where the original Gumbel trick is also useful). The presented ideas can be taken further by challenging either of these two choices:

- *Other integrals*
  The Rényi entropy is a particular integral over $\mathcal{X}$, where it so happens that one only needs to rescale the potential function in order to apply the Gumbel trick. However, (the logarithm of) an arbitrary integral $I := \sum_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x})f(\mathbf{x})$ can be rewritten as

$$\ln I = \ln \sum_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x})f(\mathbf{x}) = \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \{\ln p(\mathbf{x}) + \ln f(\mathbf{x}) + \gamma(\mathbf{x})\}\right], \tag{3.25}$$

  where $\{\gamma(\mathbf{x})\}_{\mathbf{x}\in\mathcal{X}} \overset{\text{iid}}{\sim} \text{Gumbel}(-c)$, and this can be estimated whenever the maximization inside the expectation is tractable. (Relatives of the Gumbel trick can be used to estimate transformations of I other than its logarithm.) α-Rényi entropy estimation corresponds to the special case $f(\mathbf{x}) = p(\mathbf{x})^{\alpha-1}$, which results in a simple rescaling of the potential function, thus making the maximisation tractable whenever the standard Gumbel trick would apply. An interesting avenue of future work could be exploring other classes of functions f for which the maximisation remains tractable in specific situations.

- *Other partition function estimators*
  The Rényi entropy can always be expressed as

$$H_\alpha(p) = \frac{1}{1-\alpha} \ln \sum_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x})^\alpha = \frac{1}{1-\alpha} \ln \sum_{\mathbf{x}\in\mathcal{X}} \tilde{p}(\mathbf{x})^\alpha - \frac{\alpha}{1-\alpha} \ln Z \tag{3.26}$$

$$= \frac{1}{1-\alpha} \ln Z_\alpha - \frac{\alpha}{1-\alpha} \ln Z, \tag{3.27}$$

  where $Z_\alpha := \sum_{\mathbf{x}\in\mathcal{X}} \tilde{p}(\mathbf{x})^\alpha = \sum_{\mathbf{x}\in\mathcal{X}} e^{\alpha\phi(\mathbf{x})}$ is the partition function of the distribution with potential function rescaled by α. Therefore any partition function estimation method that is capable of handling a rescaled potential function can be used to estimate the Rényi entropy using Equation (3.27). We have focused on the Gumbel trick, which (1) yields unbiased estimators in the full-rank setting and two-sided bounds in the low-rank setting, and (2) where randomness can be usefully shared between estimators of $\ln Z_\alpha$ and $\ln Z$ in order to decrease variance (Proposition 8). Exploring other partition function estimation methods and in particular studying options for sharing computation between estimators of Z_α and Z can be another interesting avenue of future work.

As with all applications of the Gumbel trick, practical usefulness is restricted to specific cases where the structure of the problem permits fast discovery of MAP solutions, or the computational complexity is dominated by a different component of the overall algorithm. This chapter presented the theoretical foundation of extending the Perturb-and-MAP framework to the integration problem of Rényi entropy estimation. An empirical evaluation of the degree to which the presented estimators are useful in various application settings is left for future work.

Chapter 4

Differentially Private Database Release

Chapter abstract In this chapter we lay theoretical foundations for new database release mechanisms that allow third-parties to construct consistent estimators of population statistics, while ensuring that the privacy of each individual contributing to the database is protected. The proposed framework rests on two main ideas. First, releasing (an estimate of) the kernel mean embedding (Smola et al., 2007) of the data generating random variable instead of the database itself still allows third-parties to construct consistent estimators of a wide class of population statistics. Second, the algorithm can satisfy the definition of differential privacy (Dwork and Roth, 2014) by basing the released kernel mean embedding on entirely synthetic data points, while controlling accuracy through the metric available in a Reproducing Kernel Hilbert Space (RKHS). Computationally, the framework leads to a class of optimization problems, where synthetic data points are optimized to minimize an RKHS distance. We describe two instantiations of the proposed framework, suitable under different scenarios, and prove theoretical results guaranteeing differential privacy of the resulting algorithms and the consistency of estimators constructed from their outputs.

4.1 Introduction

With the work presented in this chapter we aim to contribute to the body of research on the trade-off between releasing datasets from which publicly beneficial statistical inferences can be drawn, and between protecting the privacy of individuals who contribute to such datasets. Currently the most successful formalisation of protecting

user privacy is provided by *differential privacy* (Dwork and Roth, 2014), which is a *definition* that any algorithm operating on a database may or may not satisfy. An algorithm that does satisfy the definition ensures that a particular individual does not lose too much privacy by deciding to contribute to the database on which the algorithm operates.

While differentially private algorithms for releasing entire databases have been studied previously (Blum et al., 2008; Wasserman and Zhou, 2010; Zhou et al., 2009), most algorithms focus on releasing a privacy-protected version of a particular summary statistic, or of a statistical model trained on the private dataset. In this work we revisit the more difficult *non-interactive*, or *offline* setting, where the database owner aims to release a privacy-protected version of the entire database without knowing what statistics third-parties may wish to compute in the future.

In our new framework we propose to use the kernel mean embedding (Smola et al., 2007) as an intermediate representation of a database. It is (1) sufficiently rich in the sense that it captures a wide class of statistical properties of the data, while at the same time (2) it lives in a Reproducing Kernel Hilbert Space (RKHS), where it can be handled mathematically in a principled way and privacy-protected in a unified manner, independently of the type of data appearing in the database. Although kernel mean embeddings are functions in an abstract Hilbert space, in practice they can be (at least approximately) represented using a possibly weighted set of data points in input space (i.e. a set of database rows). The privacy-protected kernel mean embedding is released to the public in this representation, however, using synthetic datapoints instead of the private ones. As a result, our framework can be seen as leading to *synthetic database* algorithms.

Since the kernel mean embedding is a mathematical representation of the entire private database and it can be privacy-protected by suitable random perturbation within an RKHS, our framework shifts the computational burden from *sampling* a synthetic database (Blum et al., 2008) to the *optimization* problem of finding synthetic datapoints that minimize the RKHS distance to the perturbed kernel mean embedding. This problem transformation is useful because in specific settings the optimization can be easier to solve, and in general it possesses the *anytime* property whereby computation time and accuracy can be traded off while the differential privacy guarantee is always preserved.

We validate our approach by instantiating two concrete algorithms and proving that they output consistent estimators of the true kernel mean embedding of the

data generating process, while satisfying the definition of differential privacy. The consistency results ensure that third-parties can carry out a wide variety of statistically founded computation on the released data, such as constructing consistent estimators of population statistics, estimating the Maximum Mean Discrepancy (MMD) between distributions, and two-sample testing (Gretton et al., 2012), or using the data in the kernel probabilistic programming framework for random variable arithmetics (Schölkopf et al., 2015; Simon-Gabriel et al., 2016, Section 3), repeatedly and unlimitedly without being able to, or having to worry about, violating user privacy.

One of our algorithms is especially suited to the interesting scenario where a (small) subset of a database has already been published, a setting also considered by Ji and Elkan (2013). This situation can arise in a wide variety of settings, for example, due to weaker privacy protections in the past, due to a leak, or due to the presence of an incentive, financial or otherwise, for users to publish their data. In such a situation our algorithm provides a principled approach for reweighting the public data in such a way that the accuracy of statistical inferences on this dataset benefits from the larger sample size (including the private data), while maintaining differential privacy for the undisclosed data. Moreover, the optimization problem of minimizing the RKHS distance (here by tweaking the datapoint weights) can be solved analytically in this case.

In summary, the contributions of this chapter are:

1. A new framework for designing database release algorithms with the guarantee of differential privacy. The framework uses kernel mean embeddings as intermediate database representations, so that the RKHS metric can be used to control accuracy of the released synthetic database in a principled manner (Section 4.3).

2. Two instantiations of our framework in the form of two synthetic database algorithms, with proofs of their consistency, convergence rates and differential privacy, as well as basic empirical illustrations of their performance on synthetic datasets (Sections 4.4 and 4.5).

4.2 Background

4.2.1 Differential Privacy

Definition 2 (Dwork, 2006)**.** For $\varepsilon > 0$, $\delta \geq 0$, an algorithm $\mathcal{A}$ is said to be (ε, δ)-differentially private if for all neighbouring databases $D \sim D'$ (differing in at most one

element) and all measurable subsets S of the co-domain of $\mathcal{A}$,

$$\mathbb{P}\left(\mathcal{A}(D) \in S\right) \leq e^{\varepsilon}\mathbb{P}\left(\mathcal{A}(D') \in S\right) + \delta. \tag{4.1}$$

The parameter ε controls the amount of information the algorithm can leak about an individual, while a positive δ allows the algorithm to produce an unlikely output that leaks more information, but only with probability up to δ. This notion is sometimes called *approximate* differential privacy; an algorithm that is $(\varepsilon, 0)$-differentially private is simply said to be ε-differentially private. Note that any non-trivial differentially private algorithm must be randomised; the definition asserts that the distribution of algorithm outputs is not too sensitive to changing one row in the database.

When the algorithm's output is a finite vector $\mathcal{A}(D) \in \mathbb{R}^J$, two standard random perturbation mechanisms for making this output differentially private are the *Laplace* and *Gaussian* mechanisms. As the perturbation needs to mask the contribution of each individual entry of the database D, the scale of the added noise is closely linked to the notion of *sensitivity*, measuring how much the algorithm's output can change due to changing a single data point:

$$\Delta_1 := \sup_{D \sim D'} \left\|\mathcal{A}(D) - \mathcal{A}(D')\right\|_1, \tag{4.2}$$

$$\Delta_2 := \sup_{D \sim D'} \left\|\mathcal{A}(D) - \mathcal{A}(D')\right\|_2. \tag{4.3}$$

The Laplace mechanism adds i.i.d. $\mathrm{Lap}(\Delta_1/\varepsilon)$ noise to each of the J coordinates of the output vector and ensures pure ε-differential privacy, while the Gaussian mechanism adds i.i.d. $\mathcal{N}(0, \sigma^2)$ noise to each coordinate, where $\sigma^2 > 2\Delta_2^2 \ln(1.25/\delta)/\varepsilon^2$, and ensures (ε, δ)-differential privacy. Applying these mechanisms thus requires computing (an upper bound on) the relevant sensitivity.

Differential privacy is preserved under post-processing: if an algorithm $\mathcal{A}$ is (ε, δ)-differentially private, then so is its sequential composition $\mathcal{B}(\mathcal{A}(\cdot))$ with any other algorithm $\mathcal{B}$ that does not have direct or indirect access to the private database D (Dwork and Roth, 2014).

4.2.2 Kernels, RKHS, and Kernel Mean Embeddings

A kernel on a non-empty set (data type) $\mathcal{X}$ is a binary symmetric positive-definite function $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Intuitively it can be thought of as expressing the similarity between any two elements in $\mathcal{X}$. The literature on kernels is vast and their

properties are well studied (Schölkopf and Smola, 2002); many kernels are known for a large variety of data types such as vectors, strings, time series, graphs, etc, and kernels can be composed to yield valid kernels for composite data types (e.g. the type of a database row containing both numerical and string data).

The *kernel mean embedding* (KME) of an $\mathcal{X}$-valued random variable X in the RKHS is the function $\mu_X^k : \mathcal{X} \to \mathbb{R}$, $y \mapsto \mathbb{E}_X[k(X, y)]$, defined whenever $\mathbb{E}_X[\sqrt{k(X,X)}] < \infty$ (Smola et al., 2007). Several popular kernels have been proved to be *characteristic* (Fukumizu et al., 2008), in which case the map $p_X \mapsto \mu_X^k$, where p_X is the distribution of X, is injective. This means that no information about the distribution of X is lost when passing to its KME μ_X^k.

In practice, the KME of a random variable X is approximated using a sample $x_1, \ldots, x_N$ drawn from X, which can be used to construct an *empirical KME* $\hat{\mu}_X^k$ of X in the RKHS: a function given by $y \mapsto \frac{1}{N}\sum_{n=1}^N k(x_n, y)$. When the x_n's are i.i.d., under a boundedness condition $\hat{\mu}_X^k$ converges to the true KME μ_X^k at rate $\mathcal{O}_p(N^{-1/2})$, independently of the dimension of $\mathcal{X}$ (Lopez-Paz et al., 2015)[1]. Our approach relies on the metric of the RKHS in which these KMEs live. The RKHS $\mathcal{H}_k$ is a space of functions, endowed with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_k}$ that satisfies the *reproducing* property $\langle k(x, \cdot), h \rangle = h(x)$ for all $x \in \mathcal{X}$ and $h \in \mathcal{H}_k$. The inner product induces a norm $\| \cdot \|_{\mathcal{H}_k}$, which can be used to measure distances $\|\mu_X^k - \mu_Y^k\|_{\mathcal{H}_k}$ between distributions of X and Y. This can be exploited for various purposes such as two-sample tests (Gretton et al., 2012), independence testing (Gretton et al., 2005), or one can attempt to minimise this distance in order to match one distribution to another.

An example of such minimisation are *reduced set methods* (Burges, 1996; Schölkopf and Smola, 2002, Chapter 18), which replace a set of points $S = \{x_1, \ldots, x_N\} \subseteq \mathcal{X}$ with a weighted set $R = \{(z_1, w_1), \ldots, (z_M, w_M)\} \subseteq \mathcal{X} \times \mathbb{R}$ (of potentially smaller size), where the new points z_m can, but need not equal any of the x_ns, such that the KME computed using the reduced set R is close to the KME computed using the original set S, as measured by the RKHS norm:

$$\left\|\mu_S^k - \mu_R^k\right\|_{\mathcal{H}_k} = \left\|\frac{1}{N}\sum_{n=1}^N k(x_n, \cdot) - \sum_{m=1}^M w_m k(z_m, \cdot)\right\|_{\mathcal{H}_k}.$$

[1]The KME can be viewed as a smoothed version of the density, which is easier to estimate than the density itself; rates of nonparametric density estimation or statistical powers of two-sample or independence tests involving p_X are known to necessarily degrade with growing dimension (Tolstikhin et al., 2017, Section 4.3).

Reduced set methods are usually motivated by the computational savings arising when $|R| < |S|$; we will invoke them mainly to replace a collection S of private data points with a (possibly weighted) set R of synthetic data points.

4.3 Framework

4.3.1 Problem Formulation

Throughout this chapter, we assume the following setup. A database curator wishes to publicly release a database $D = \{x_1, \dots x_N\} \in \mathcal{X}^N$ containing private data about N individuals, with each data point (database row) x_n taking values in a non-empty set $\mathcal{X}$. The set $\mathcal{X}$ can be arbitrarily rich, for example, it could be a product of Euclidean spaces, integer spaces, sets of strings, etc.; we only require availability of a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ on $\mathcal{X}$. We assume that the N rows $x_1, \dots, x_N$ in the database D can be thought of as i.i.d. observations from some $\mathcal{X}$-valued data-generating random variable X (but see Section 4.7 for a discussion about relaxing this assumption). The database curator, wishing to protect the privacy of individuals in the database, seeks a database release mechanism that satisfies the definition of (ε, δ)-differential privacy, with $\varepsilon > 0$ and $\delta \geq 0$ given. The main purpose of releasing the database is to allow third parties to construct estimators of population statistics (i.e. properties of the distribution of X), but it is not known at the time of release what statistics the third-parties will be interested in.

To lighten notation, henceforth we drop the superscript k from KMEs (such as μ_X^k) and the subscript k from the RKHS $\mathcal{H}_k$, whenever k is the kernel on $\mathcal{X}$ chosen by the database curator.

4.3.2 Algorithm Template

We propose the following general algorithm template for differentially private database release:

1. Construct a consistent estimator $\hat{\mu}_X$ of the KME μ_X of X using the private database.

2. Obtain a perturbed version $\tilde{\mu}_X$ of the constructed estimate $\hat{\mu}_X$ to ensure differential privacy.

3. Release a (potentially approximate) representation of $\tilde{\mu}_X$ in terms of a (possibly weighted) dataset $\{(z_1, w_1), \ldots, (z_M, w_M)\} \subseteq \mathcal{X} \times \mathbb{R}$.

The released representation should be such that $\sum_{m=1}^{M} w_m k(z_m, \cdot)$ is a consistent estimator of the true KME μ_X, i.e. such that the RKHS distance between the two converges to 0 in probability as the private database size N, and together with it the synthetic database size M, go to infinity.

Each step of this template admits several possibilities. For the first step we have discussed the standard empirical KME $\frac{1}{N}\sum_{n=1}^{N} k(x_n, \cdot)$ with $x_1, \ldots, x_N$ i.i.d. observations of X, but the framework remains valid with improved estimators such as *kernel-based quadrature* (Chen et al., 2010) or the *shrinkage* kernel mean estimators of (Muandet et al., 2016).

As the KMEs $\hat{\mu}_X$ and μ_X live in the RKHS $\mathcal{H}$ of the kernel k, a natural mechanism for privatising $\hat{\mu}_X$ in the second step would be to follow (Hall et al., 2013) and add pointwise to $\hat{\mu}_X$ a suitably scaled sample path g of a Gaussian process with covariance function $k(\cdot, \cdot)$. This does ensure (ε, δ)-differential privacy of the resulting function $\tilde{\mu}_X = \hat{\mu}_X + g$, but unfortunately $\tilde{\mu}_X \notin \mathcal{H}$, because the RKHS norm $\|g\|_{\mathcal{H}}$ of a Gaussian process sample path with the same kernel k is infinite almost surely (Rasmussen and Williams, 2005). While our framework allows pursuing this direction by, for example, moving to a larger function space that does contain the Gaussian process sample path, in this work we will present algorithms that achieve differential privacy by mapping $\hat{\mu}_X$ into a finite-dimensional Hilbert space and then employing the standard Laplace or Gaussian mechanisms on the finite coordinate vector.

Differential privacy is preserved under post-processing, but the third step does require some care to ensure that private data is not leaked. Specifically, when several possible (approximate) representations $\tilde{\mu}_X \approx \sum_{m=1}^{M} w_m k(z_m, \cdot)$ in terms of a weighted dataset $(w_1, z_1), \ldots, (w_M, z_M)$ are possible, committing to a particular one reveals more information than just the function $\tilde{\mu}_X$ (consider, for example, the extreme case where the representation would be in terms of the private points $x_1, \ldots, x_N$). One thus needs to either control the privacy leak due to choosing a representation in a way that depends on the private data, or, as we do in our concrete algorithms below, choose a representation independently of the private data (but still minimising its RKHS distance to the privacy-protected $\tilde{\mu}_X$).

4.3.3 Versatility

Algorithms in our framework release a possibly weighted synthetic dataset

$$\{(z_1, w_1), \ldots, (z_M, w_M)\} \subseteq \mathcal{X} \times \mathbb{R} \tag{4.4}$$

such that $\sum_{m=1}^{M} w_m k(z_m, \cdot)$ is a consistent estimator of the true KME μ_X of the data generating random variable X. This allows third-parties to perform a wide spectrum of statistical computation, all without having to worry about violating differential privacy:

1. *Kernel probabilistic programming* (Schölkopf et al., 2015): The versatility of our approach is greatly expanded thanks to the result of (Simon-Gabriel et al., 2016), who showed that under technical conditions, applying a continuous function f to all points z_m in the synthetic dataset yields a consistent estimator $\sum_{m=1}^{M} w_m k_f(f(z_m), \cdot)$ of the KME $\mu_{f(X)}$ of the transformed random variable $f(X)$, even when the points $z_1, \ldots, z_M$ are not i.i.d. (as they may not be, depending on the particular synthetic database release algorithm).

2. *Consistent estimation of population statistics*: For any RKHS function $h \in \mathcal{H}$, we have $\langle \mu_X, h \rangle_{\mathcal{H}} = \mathbb{E}[h(X)]$, so a consistent estimator of μ_X yields a consistent estimator of the expectation of $h(X)$. It can be evaluated using the reproducing kernel property:

$$\mathbb{E}[h(X)] = \langle \mu_X, h \rangle_{\mathcal{H}} \approx \left\langle \sum_{m=1}^{M} w_m k(z_m, \cdot), h \right\rangle_{\mathcal{H}} = \sum_{m=1}^{M} w_m h(z_m). \tag{4.5}$$

   For example, approximating the indicator function $\mathbb{1}_S$ of a set $S \subseteq \mathcal{X}$ with functions in the RKHS allows estimating probabilities: $\mathbb{E}[\mathbb{1}_S(X)] = \mathbb{P}[X \in S]$ (note that $\mathbb{1}_S$ itself may not be an element of the RKHS).

3. *MMD estimation and two-sample testing* (Gretton et al., 2012): Given another random variable Y on $\mathcal{X}$, one can consistently estimate the Maximum Mean Discrepancy (MMD) distance $\|\mu_X - \mu_Y\|_{\mathcal{H}}$ between the distributions of X and Y, and in particular construct a two-sample test based on this distance. Given a sample $y_1, \ldots, y_L \sim Y$:

$$\|\mu_X - \mu_Y\|_{\mathcal{H}} \approx \left\| \sum_{m=1}^{M} w_m k(z_m, \cdot) - \frac{1}{L} \sum_{l=1}^{L} k(y_l, \cdot) \right\|_{\mathcal{H}}, \tag{4.6}$$

which can again be evaluated using the reproducing property.

4. *Subsequent use of synthetic data*: Since the output of the algorithm is a (possibly weighted) database, third-parties are free to use this data for arbitrary purposes, such as training any machine learning model on this data. Models trained purely on this data can be released with differential privacy guaranteed; however, the accuracy of such models on real data remains an empirical question that is beyond the scope of this work.

An orthogonal spectrum of versatility arises from the fact that the third step in the algorithm template can constrain the released dataset $(z_1, w_1), \ldots, (z_M, w_M)$ to be more convenient or more computationally efficient for further processing. For example, one could fix the weights to uniform $w_m = \frac{1}{M}$ to obtain an unweighted dataset, or to replace an expensive data type with a cheaper subset, such as requesting floats instead of doubles in the z_m's. All this can be performed while an RKHS distance is available to control accuracy between $\tilde{\mu}_X$ and its released representation.

4.3.4 Concrete Algorithms

As a first illustrative example, we describe how a particular case of an existing, but inefficient synthetic database algorithm already fits into our framework. The *exponential mechanism* (McSherry and Talwar, 2007) is a general mechanism for ensuring ε-differential privacy, and in our setting it operates as follows: given a similarity measure $s : \mathcal{X}^N \times \mathcal{X}^M \to \mathbb{R}$ between (private) databases of size N and (synthetic) databases of size M, output a random (synthetic) database R with probability proportional to $\exp(\frac{\varepsilon}{2\Delta_1} s(D, R))$, where D is the actual private database and Δ_1 is the L_1 sensitivity of s w.r.t. D. This ensures ε-differential privacy (McSherry and Talwar, 2007).

This instantiation of the exponential mechanism already fits into our framework by taking $s(D, R) = -\|\mu_D - \mu_R\|_{\mathcal{H}}$ to be the negative RKHS distance between the KMEs computed using D and R, and ε-differential privacy is achieved by releasing R with probability proportional to $\exp(-\frac{\varepsilon}{2\Delta_1}\|\mu_D - \mu_R\|_{\mathcal{H}})$. This procedure solves steps 2 and 3 of our general algorithm template simultaneously, as it directly samples a concrete representation of a "perturbed" KME μ_R. The algorithm essentially corresponds to the SmallDB algorithm of Blum et al. (2008), except for choosing the RKHS distance as a well-studied similarity measure between two databases.

The principal issue with this algorithm is its computational infeasibility except in trivial cases, as it requires sampling from a probability distribution supported on all

potential synthetic databases, and employing an approximate sampling scheme can break the differential privacy guarantee of the exponential mechanism. In Sections 4.4 and 4.5 respectively, we describe two concrete synthetic database release algorithms that may possess failure modes where they become inefficient, but employing approximations in those cases can only affect their statistical accuracy, not the promise of differential privacy.

Algorithm 2 Differentially private database release via a synthetic data subspace

Input: database $D = \{x_1, \ldots, x_N\} \subseteq \mathcal{X}$, kernel k on $\mathcal{X}$, privacy specs $\varepsilon > 0$, $\delta > 0$
Output: (ε, δ)-differentially private, weighted synthetic database (estimating μ_X)

$M \leftarrow M(N) \in \omega(1) \cap o(N^2)$, number of synthetic data points to use
$z_1, \ldots, z_M \leftarrow$ initialise deterministically or randomly from a distribution q on $\mathcal{X}$
$\mathcal{H}_M \leftarrow \mathrm{Span}(\{k(z_1, \cdot), \ldots, k(z_M, \cdot)\}) \leq \mathcal{H}$
$b_1, \ldots, b_F \leftarrow$ orthonormal basis of $\mathcal{H}_M$ (obtained using the Gram-Schmidt process)

$\hat{\mu}_X \leftarrow \frac{1}{N} \sum_{n=1}^{N} k(x_n, \cdot)$, empirical KME of X in $\mathcal{H}$
$\overline{\mu}_X \leftarrow \sum_{f=1}^{F} \langle b_f, \hat{\mu}_X \rangle_{\mathcal{H}} b_f =: \sum_{f=1}^{F} \alpha_f b_f$, projection of $\hat{\mu}_X$ onto $\mathcal{H}_M$
$\boldsymbol{\beta} \leftarrow \boldsymbol{\alpha} + \mathcal{N}(0, \frac{8 \ln(1.25/\delta)}{N^2 \varepsilon^2} I_F)$, an (ε, δ)-DP version of coordinates $\boldsymbol{\alpha}$ (Gaussian mech.)

$\tilde{\mu}_X \leftarrow \sum_{f=1}^{F} \beta_f b_f = \sum_{m=1}^{M} w_m k(z_m, \cdot)$, re-expressed in terms of $k(z_m, \cdot)$'s
return $(z_1, w_1), \ldots, (z_M, w_M)$

4.4 Perturbation in Synthetic-Data Subspace

In this section we describe an instantiation of the framework proposed in Section 4.3 that achieves differential privacy of the KME by projecting it onto a finite-dimensional subspace of the RKHS spanned by feature maps $k(z_m, \cdot)$ of synthetic data points $z_1, \ldots, z_M$, and perturbing the resulting finite coordinate vector. To ensure differential privacy, the synthetic data points are chosen independently of the private database. As a result, statistical efficiency of this approach will depend on the choice of synthetic data points, with efficiency increasing if there are enough synthetic data points to capture the patterns in the private data. Therefore this algorithm is especially suited to the scenario discussed in Section 4.1, where some part of the database (or of a similar one) has already been published, as this can serve as a good starting set for the synthetic data points.

The setting where some observations from X have already been released highlights the fact that differential privacy only protects against *additional* privacy violation due

to an individual deciding to contribute to the private database; if a particular user's data has already been published, differential privacy does not protect against privacy violations based on exploiting this previously published data.

The algorithm is formalised as Algorithm 2 above. Lines 1-2 choose synthetic data points $z_1, \ldots, z_M$ independently of the private data (only using the database size N). Lines 3-4 construct the linear subspace $\mathcal{H}_M$ of $\mathcal{H}$ spanned by feature maps of the chosen synthetic data points, and compute a (finite) basis for it. Only then the private data is accessed: the empirical KME $\hat{\mu}_X$ is computed (line 5), projected onto the subspace $\mathcal{H}_M$ and expressed in terms of the precomputed basis (line 6). The basis coefficients of the projection are then perturbed using the Gaussian mechanism to achieve differential privacy (line 7), and the perturbed element $\tilde{\mu}_X \in \mathcal{H}_M$ is then re-expressed in terms of the spanning set containing feature maps of synthetic data points (line 8). This expansion is finally released to the public (line 9).

Line 1 stipulates that the number of synthetic data points $M \to \infty$ as $N \to \infty$, but asymptotically slower than N^2. This is to ensure that the privatisation noise added in the subspace $\mathcal{H}_M$ to each coordinate is small enough overall to preserve consistency, as stated in the following Theorem 10. This theorem assures us that Algorithm 2 produces a consistent estimator of the true KME μ_X, if the synthetic data points are sampled from a distribution with sufficiently large support. All proofs appear in Appendix C.1.

Theorem 10. *Let $\mathcal{X}$ be a compact metric space and k a continuous kernel on $\mathcal{X}$. If the synthetic data points $z_1, z_2, \ldots$ are sampled i.i.d. from a distribution q on $\mathcal{X}$ such that the support of X is included in the support of q, then Algorithm 2 outputs a consistent estimator of the KME μ_X, i.e. $\sum_{m=1}^{M} w_m k(z_m, \cdot) \xrightarrow{\mathbb{P}} \mu_X$ as $N \to \infty$.*

As discussed by Simon-Gabriel et al. (2016), these assumptions are usually satisfied: $\mathcal{X}$ can be taken to be compact whenever the data comes from measurements with any bounded range, and many kernels are continuous, including all kernels on discrete spaces (w.r.t. to the discrete topology).

In order to use the output of Algorithm 2 in the very general *kernel probabilistic programming* framework and obtain a consistent estimator of the KME $\mu_{f(X)}$ of $f(X)$ for *any* continuous function f, there is a technical condition that the L_1 norm $\sum_{m=1}^{M} |w_m|$ of the released weights may need to remain bounded by a constant as $N \to \infty$ (Simon-Gabriel et al., 2016). This is not enforced by Algorithm 2, but Theorem 13 in Appendix C.1.1 shows how a simple regularisation in the final stage of the algorithm achieves this without breaking consistency (or privacy).

The next result about Algorithm 2 shows that it is differentially private whenever $k(x, x) \leq 1$ for all $x \in \mathcal{X}$. This is a weak assumption that holds for all normalised kernels, and can be achieved by simple rescaling for any bounded kernel (such that $\sup_{x \in \mathcal{X}} k(x, x) < \infty$). When $\mathcal{X}$ is a compact domain, all continuous kernels are bounded.

Proposition 10. *If $k(x, x) \leq 1$ for all $x \in \mathcal{X}$, then Algorithm 2 is (ε, δ)-differentially private.*

Remark. One usually requires that δ decreases faster than polynomially with the database size N (Dwork and Roth, 2014). The proof of Theorem 10 remains valid whenever $M(N) \in o(N^2 / \ln(1.25/\delta(N)))$, so for example we could have $\delta(N) = e^{-\sqrt{N}}$ and $M(N) \in o(N^{3/2})$. □

For a finite private database, actual performance will heavily depend on how the synthetic data points are chosen. We consider the following two extreme scenarios:

1. *No publishable subset*: No rows of the private database are, or can be made public unmodified.

2. *Publishable subset*: A small proportion of the private database is already public, or can be made public.

Proposition 11 (Algorithm 1, No publishable subset)**.** *Say $\mathcal{X}$ is a bounded subset of $\mathbb{R}^D$, the kernel k is Lipschitz, and the synthetic data points $z_1, z_2, \ldots$ are sampled i.i.d. from a distribution q with density bounded away from 0 on any bounded subset of $\mathbb{R}^D$. Then $M = M(N)$ can be chosen so that the output of Algorithm 2 converges to the true KME μ_X in RKHS norm at a rate $\mathcal{O}_p(N^{-1/(D+1+c)})$, where c is any fixed positive number $c > 0$.*

Unsurprisingly, the convergence rate deteriorates with input dimension D, since without prior information about the private data manifold it is increasingly difficult for randomly sampled synthetic points to capture patterns in the private data. One of the main strengths of KMEs is that the empirical estimator converges to the true embedding at a rate $\mathcal{O}_p(N^{-1/2})$ independently of the input dimension D, so we see that in this unfavourable scenario Algorithm 2 incurs a substantial privacy cost in high dimensions. On the other hand, if a small, but fixed proportion of the private database is publishable, then Algorithm 2 incurs no privacy cost in terms of the convergence rate:

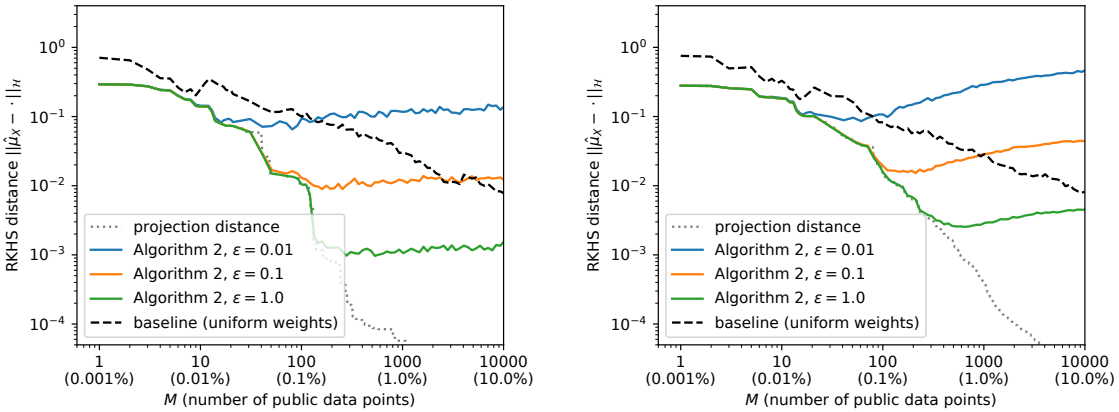


Fig. 4.1 RKHS distance (lower is better) to the (private) empirical KME $\hat{\mu}_X$ computed using the entire private database of size $N = 100,000$. The dimension of the database was $D = 2$ (left) or $D = 5$ (right); please see Appendix C.2 for further details of the setup. Horizontally we varied M, the number of publicly releasable data points. Stricter privacy requirements (lower ε) naturally lead to lower accuracy. Increasing M does not always necessarily improve accuracy, since a new public data point always increases the total amount of privatising noise that needs to be added, but this might not be outweighed by its positive contribution towards covering relevant parts of the input space. In all cases, for sufficiently small M Algorithm 2 provided a more accurate estimate than μ^{baseline}.

Proposition 12 (Algorithm 1, Publishable subset)**.** *Say that a fixed proportion η of the private database can be published unmodified. Using this part of the database as the synthetic data points, Algorithm 2 outputs a consistent estimator of μ_X that converges in RKHS norm at a rate $\mathcal{O}_p(N^{-1/2})$.*

Note that in this scenario differential privacy and the rate $\mathcal{O}_p(N^{-1/2})$ can be also achieved by uniform weighting of the synthetic data points, as $\hat{\mu}^{\text{baseline}} := \frac{1}{M}\sum_{m=1}^{M} k(z_m, \cdot)$ with $z_m = x_m$ is already a consistent estimator of μ_X (although based on a much smaller sample size $M = \eta N \ll N$). The purpose of Algorithm 2 is to find (generally non-uniform) $w_1, \ldots, w_M$ that reweight the public data points using the information in the large private dataset, but respecting differential privacy. Proposition 12 confirmed theoretically that this does not hurt the convergence rate, while Figure 4.1 shows empirically on two synthetic datasets of dimensions $D = 2$ and $D = 5$ that Algorithm 2 can in fact yield more accurate estimates of the KME than $\hat{\mu}^{\text{baseline}}$, especially when the proportion of public data points is small. This is encouraging, since obtaining permission to publish a larger subset of the private data unchanged will usually come at an increased cost. The ability to instead reweight a smaller public dataset in a differentially private manner using Algorithm 2 is therefore useful.

Algorithm 3 Differentially private database release via a random features RKHS

Input: database $D = \{x_1, \ldots, x_N\} \subseteq \mathcal{X}$, kernel k on $\mathcal{X}$, privacy specs $\varepsilon > 0$, $\delta > 0$
Output: (ε, δ)-differentially private, weighted synthetic database (estimating μ_X)

1: $J \leftarrow J(N) \in \omega(1) \cap o(N^2)$, number of random features to use
2: $\phi \leftarrow$ random feature map $\mathcal{X} \mapsto \mathbb{R}^J$ for kernel k with J features
3: $\hat{\mu}_X^\phi \leftarrow \frac{1}{N}\sum_{n=1}^N \phi(x_n) \in \mathbb{R}^J$, empirical KME of X in the RKHS $\mathcal{H}_\phi$ of the random features kernel $k_\phi(\cdot,\cdot) := \phi(\cdot)^T\phi(\cdot)$
4: $\tilde{\mu}_X^\phi \leftarrow \hat{\mu}_X^\phi + \mathcal{N}(0, \frac{8\ln(1.25/\delta)}{N^2\varepsilon^2} I_J)$, an (ε, δ)-DP version of $\hat{\mu}_X^\phi$ (Gaussian mechanism)
5: $M \leftarrow M(N) \geq N$, number of synthetic expansion points to use for representing $\tilde{\mu}_X^\phi$
6: $(z_1, w_1), \ldots, (z_M, w_M) \leftarrow$ approximate $\tilde{\mu}_X^\phi$ in $\mathcal{H}_\phi$ using a Reduced set method:

$$(z_1, w_1), \ldots, (z_M, w_M) \approx \underset{(z'_1,w'_1),\ldots,(z'_M,w'_M) \text{ s.t. } \sum_m |w'_m| \leq 1}{\operatorname{argmin}} \left\| \sum_{m=1}^M w'_m \phi(z'_m) - \tilde{\mu}_X^\phi \right\|_{\mathcal{H}_\phi} \tag{4.7}$$

7: **return** $(z_1, w_1), \ldots, (z_M, w_M)$

4.5 Perturbation in Random-Features RKHS

Another approach to ensuring differential privacy is to map the potentially infinite dimensional RKHS $\mathcal{H}$ of k into a different, finite-dimensional RKHS $\mathcal{H}_\phi$ using random features (Rahimi and Recht, 2007), privacy-protect the finite coordinate vector in this space (Chaudhuri et al., 2011), and then employ a reduced set method to find an expansion of the resulting RKHS element in terms of synthetic data points. In contrast to Algorithm 2, both the weights and locations of synthetic data points can be optimised here.

The algorithm is formalised as Algorithm 3. Lines 1-2 pick the number $J = J(N)$ of random features to use, and construct a random feature map ϕ with that many features. Lines 3-4 compute the empirical KME of X in the RKHS $\mathcal{H}_\phi$ corresponding to the kernel induced by the random features, and then privacy-protect the resulting finite, real-valued vector using the Gaussian mechanism. Lines 5-6 run a blindly initialised Reduced set method to find a weighted synthetic dataset whose KME in $\mathcal{H}_\phi$ is close to the privacy-protected KME of the private database. Finally, line 7 releases this weighted dataset to the public.

The following theorem confirms that Algorithm 3 outputs a consistent estimator of the true KME μ_X, provided that the optimisation problem (4.7) is solved exactly, and the random features converge to the kernel k uniformly on $\mathcal{X}$. On compact sets $\mathcal{X}$ this requirement is satisfied by general schemes such as *random Fourier features* and

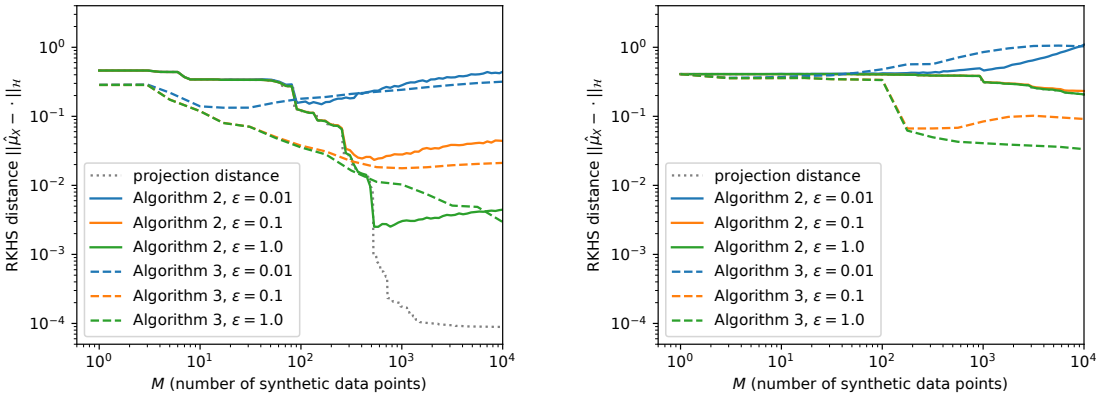


Fig. 4.2 RKHS distance (lower is better) to the (private) empirical KME $\hat{\mu}_X$ computed using the same databases as in Figure 4.1, of dimensions $D = 2$ (left) and $D = 5$ (right), but this time without a publishable subset. The synthetic data points for Algorithm 2 were therefore sampled from a wide Gaussian distribution; please see Appendix C.2 for further details. Algorithm 3 is capable of outperforming Algorithm 2 thanks to its ability to optimise the synthetic data point locations, but this depends on the precise optimisation procedure used and the optimisation problem becomes harder in higher dimensions.

random binning for shift-invariant kernels (Rahimi and Recht, 2007), schemes such as the Mondrian kernel (Balog et al., 2016), or by random features for dot product kernels (Kar and Karnick, 2012).

Theorem 11. *If $\phi(\cdot)^T\phi(\cdot) \to k(\cdot,\cdot)$ converges uniformly in $\mathcal{X} \times \mathcal{X}$ as $J \to \infty$, then the output of Algorithm 3 is a consistent estimator of the true KME μ_X as $N \to \infty$.*

Moreover, a uniform convergence rate for the random features, such as the one for random Fourier features by Sriperumbudur and Szabo (2015), can be used to derive a convergence rate for the output of Algorithm 3:

Proposition 13. *If $\phi(\cdot)^T\phi(\cdot) \to k(\cdot,\cdot)$ converges uniformly in $\mathcal{X} \times \mathcal{X}$ at a rate $\mathcal{O}_p(J^{-1/2})$ as $J \to \infty$, then $J = J(N)$ can be chosen so that the output of Algorithm 3 converges to the true KME μ_X at a rate $\mathcal{O}_p(N^{-1/3})$.*

The empirical KME of the private database $\hat{\mu}_X$ converges at a rate $\mathcal{O}_p(N^{-1/2})$, so we see that under perfect optimisation, the privacy cost incurred by Algorithm 3 is a factor of $N^{1/6}$. In practice performance will also depend on the Reduced set method used, and the computational budget allocated to it. Figure 4.2 shows how the incurred error (in terms of RKHS distance) varies with the number of synthetic data points M. The additional ability of Algorithm 3 to optimise the *locations* of the synthetic data points (rather than just the weights, as in Algorithm 2) seems to be more helpful in

the higher-dimensional case $D = 5$, where the randomly sampled synthetic data points are less likely to land close to private data points.

However, the problem gets harder in higher dimensions, because of the combined effect of 1) requiring more noise to achieve differential privacy, and 2) the optimization landscape becoming more difficult.

Proposition 14. *If $\|\phi(x)\|_2 \leq 1$ for all $x \in \mathcal{X}$, then Algorithm 3 is (ε, δ)-differentially private.*

This L_2-boundedness requirement on the random feature vectors $\phi(x)$ is reasonable under the weak assumption $k(x, x) \leq 1$ for all $x \in \mathcal{X}$ discussed in Section 4.4, as in that case $\|\phi(x)\|_2^2 = \phi(x)^T \phi(x) \approx k(x, x) \leq 1$.

4.6 Related Work

Synthetic database release algorithms with a differential privacy guarantee have been studied in the literature before. Machanavajjhala et al. (2008) analyzed such a procedure for count data, ensuring privacy by sampling a distribution and then synthetic counts from a Dirichlet-Multinomial posterior. Blum et al. (2008) studied the exponential mechanism applied to synthetic database generation, which leads to a very general, but unfortunately inefficient algorithm (see also Section 4.3.4). Wasserman and Zhou (2010) provided a theoretical comparison of this algorithm to sampling synthetic databases from deterministically smoothed, or randomly perturbed histograms. Unlike our approach, these algorithms achieve differential privacy by sampling synthetic data points from a specific distribution, where resorting to approximate sampling can break the privacy guarantee. In our framework we propose to arrive at the synthetic database using a reduced set method, where poor performance could affect statistical usefulness of the synthetic database, but cannot break its differential privacy.

Zhou et al. (2009) and Kenthapadi et al. (2012) proposed randomised database compression schemes that yield synthetic databases useful for particular types of algorithms, while guaranteeing differential privacy. The former compresses the number of data points using a random linear or affine transformation of the entire database, and the result can be used by procedures that rely on the empirical covariance of the original data. The latter compresses the number of data point dimensions while approximately preserving distances between original, private data points.

More recently, Beaulieu-Jones et al. (2019) proposed to train Generative Adversarial Networks (GANs, Goodfellow et al., 2014) to generate differentially private synthetic datasets by utilizing differentially-private SGD (Abadi et al., 2016) to train the *discriminator*, the only part of the GAN that has access to the private data.

Differentially private learning in a RKHS has also been studied, with Chaudhuri et al. (2011) and Rubinstein et al. (2012) having independently presented release mechanisms for the result of an empirical risk minimisation procedure (such as a SVM). Similarly to our Algorithm 3, they map data points into a finite-dimensional space defined by random features and carry out the privacy-protecting perturbation there. However, they do not require the final stage of invoking a Reduced set method to construct a synthetic database, because the output (such as a trained SVM) is only used for evaluation on test points, for which it suffices to additionally release the used random feature map ϕ. After us, Raj et al. (2019) considered the particular task of kernel two-sample testing under the constraint of differential privacy.

As our framework stipulates privacy-protecting an empirical KME, which is a function $\mathcal{X} \to \mathbb{R}$, the work on differential privacy for functional data is of relevance. Hall et al. (2013) showed how an RKHS element can be made differentially private via pointwise addition of a Gaussian process sample path, but as discussed in Section 4.3.2, the resulting function is no longer an element of the RKHS. More recently, Aldà and Rubinstein (2017) proposed a general Bernstein mechanism for ε-differentially private function release. The released function can be evaluated pointwise arbitrarily many times, but again, the geometry of the RKHS to which the unperturbed function belonged cannot be easily exploited anymore.

4.7 Discussion and Future Work

We proposed a framework for constructing differentially private synthetic database release algorithms, based on the idea of using KMEs in RKHS as intermediate database representations that can be 1) perturbed to guarantee differential privacy, and 2) subsequently released for versatile applications (see Section 4.3.3). To justify our framework, we presented two concrete algorithms and proved theoretical results guaranteeing their consistency and differential privacy. We also studied their finite-sample convergence rates, and provided empirical illustrations of their performance on simulated datasets. We believe that exploring other instantiations of this framework, and comparing them theoretically and empirically, can be a fruitful direction for future research.

Code to reproduce experiments in this chapter is available at https://github.com/matejbalog/RKHS-private-database.

Limitations The work presented in this chapter is primarily theoretical in nature, focusing on finite-sample and asymptotic convergence rates of constructed statistical estimators under the assumption that the reduced set optimization problem in Equation 4.7 can be solved (in the case of Algorithm 3). This highlights the two main current limitations. First, while the optimization problem generated by our framework does have the promised *anytime* property whereby computation time and accuracy can be traded off while preserving differential privacy, the trade off can still be expensive. In Algorithm 2 the optimization problem could in fact be solved analytically, but in Algorithm 3 it is a highly non-convex problem that required a custom iterative gradient-based optimization procedure inspired by Schölkopf and Smola (2002, Chapter 18) to make progress on. Second, an evaluation on real-world sensitive datasets is required to assess whether the proposed concrete algorithms are already practically applicable, or more algorithmic innovation within the framework is needed.

The flexibility of the proposed framework also poses a potential limitation as it leaves multiple modelling and algorithmic choices to the practitioner. These include, for example, the choice of kernel k, the choice of random features ϕ in Algorithm 3, and in the case of applying Algorithm 2 without pre-existing public data points also the choice of the base distribution q. In these cases, part of the privacy budget ε could be allocated to choosing these hyper-parameters based on the private data.

Future directions Apart from addressing current limitations, further possible future directions are listed below.

- **Relaxations**
  The i.i.d. assumption on database rows can be relaxed. For example, if they are identically distributed (as a random variable X), but not necessarily independent, the framework remains valid as long as a consistent estimator of the KME μ_X can be constructed from the database rows. A common situation where this arises is, for example, duplication of database rows due to user error.

- Relationship to **hardness results**
  As discussed in Section 1.5.3, the offline (non-interactive) setting of differential privacy is more challenging because the final use case is not known at time of

release and therefore the privatization step cannot be specialized to a particular query. This intuition has been transformed into formal computational hardness results (Kowalczyk et al., 2018) and it would be insightful to understand how these relate to the framework proposed in this chapter. This would firstly require unifying the language of *queries* in differential privacy with the notion of *statistical computation* using estimators of kernel mean embeddings as they were discussed in Section 4.3.3.

- Perturbations in **infinite-dimensional RKHS**
  Our concrete Algorithms 2 and 3 both construct a finite-dimensional RKHS in which the random perturbation ensuring differential privacy can be carried out using the standard Laplace and Gaussian mechanisms (Dwork and Roth, 2014) on finite-dimensional coordinate vectors. As discussed in Section 4.3.2, this is motivated by the fact that while the Gaussian mechanism has been generalized to (infinite-dimensional) RKHS by Hall et al. (2013), the algorithm requires pointwise addition of a Gaussian process sample path that moves the result outside of the original RKHS (more specifically, to its external boundary where $\|\cdot\|_{\mathcal{H}} = \infty$).

  Pursuing this approach by moving to a slightly larger RKHS that does contain the Gaussian process sample path (Kanagawa et al., 2018) could lead to new algorithms that eliminate the "projection" error component (see Section C.1 in the Appendices). Alternatively, projecting the perturbed element back onto a bounded subset of the original RKHS could be considered. Depending on the choice of subset, this may or may not be equivalent to our Algorithm 2.

- Differentially private **MMD-GAN**
  Among deep generative models, adversarial training of GANs (Goodfellow et al., 2014) has been a highly successful approach for generating realistic data points, and this has recently also been exploited for generating differentially private synthetic databases (Beaulieu-Jones et al., 2019). Since kernel-based versions of GANs employing discriminators relying on the MMD metric (Gretton et al., 2012) in an RKHS have already been proposed and studied (Dziugaite et al., 2015; Li et al., 2017, 2015), a natural combination to consider could be to study differentially private versions of such MMD-GANs.

Chapter 5

Program Synthesis

Chapter abstract We develop a first line of attack for solving programming contest-style problems from input-output examples using deep learning. The approach is to cast program synthesis – a discrete search problem – as a supervised learning problem that can be attacked with gradient-based continuous optimization. We train a neural network to predict properties of the program that generated the outputs from the inputs. We use the neural network's predictions to augment search techniques from the programming languages community, including enumerative search and an SMT-based solver. Empirically, we show that our approach leads to an order of magnitude speedup over the strong non-augmented baselines and a Recurrent Neural Network approach, and that we are able to solve problems of difficulty comparable to the simplest problems on programming competition websites.

5.1 Introduction

A dream of artificial intelligence is to build systems that can write computer programs. There has been much interest in program-like neural network models (Graves et al., 2014, 2016; Grefenstette et al., 2015; Joulin and Mikolov, 2015; Kaiser and Sutskever, 2016; Kurach et al., 2016; Neelakantan et al., 2016; Reed and De Freitas, 2016; Sukhbaatar et al., 2015; Weston et al., 2015; Zaremba et al., 2016), but none of these can *write programs*; that is, they do not generate human-readable source code. Only very recently, Bošnjak et al. (2017); Bunel et al. (2016); Gaunt et al. (2016) explored the use of gradient descent to induce source code from input-output examples via differentiable interpreters, and Ling et al. (2016) explored the generation of source code from unstructured text descriptions. However, Gaunt et al. (2016) showed that

differentiable interpreter-based program induction is inferior to discrete search-based techniques used by the programming languages community. We are then left with the question of how to make progress on program induction using machine learning techniques.

In this work, we propose two main ideas: (1) learn to induce programs; that is, use a corpus of program induction problems to learn strategies that generalize across problems, and (2) integrate neural network architectures with search-based techniques rather than replace them.

In more detail, we can contrast our approach to existing work on differentiable interpreters. In differentiable interpreters, the idea is to define a differentiable mapping from source code and inputs to outputs. After observing inputs and outputs, gradient descent can be used to search for a program that matches the input-output examples. This approach leverages gradient-based optimization, which has proven powerful for training neural networks, but each synthesis problem is still solved independently—solving many synthesis problems does not help to solve the next problem.

We argue that machine learning can provide significant value towards solving Inductive Program Synthesis (IPS) by re-casting the problem as a big data problem. We show that training a neural network on a large number of generated IPS problems to predict cues from the problem description can help a search-based technique. In this work, we focus on predicting an order on the program space and show how to use it to guide search-based techniques that are common in the programming languages community. This approach has three desirable properties: first, we transform a difficult *discrete search* problem into a supervised learning problem that can be approached with *gradient-based optimization*; second, we soften the effect of failures of the neural network by searching over program space rather than relying on a single prediction; and third, the neural network's predictions are used to guide existing program synthesis systems, allowing us to use and improve on the best solvers from the programming languages community. Empirically, we show orders-of-magnitude improvements over optimized standard search techniques and a Recurrent Neural Network-based approach to the problem.

In summary, we define and instantiate a framework for using deep learning for program synthesis problems like ones appearing on programming competition websites. The concrete contributions are:

1. defining a programming language that is expressive enough to include real-world programming problems while being high-level enough to be predictable from input-output examples;

2. models for mapping sets of input-output examples to program properties; and

3. experiments that show an order of magnitude speedup over standard program synthesis techniques, which makes this approach feasible for solving problems of similar difficulty as the simplest problems that appear on programming competition websites.

5.2 Background

We begin by providing background on Inductive Program Synthesis, including a brief overview of how it is typically formulated and solved in the programming languages community.

The *Inductive Program Synthesis* (IPS) problem is the following: given input-output examples, produce a program that has behavior consistent with the examples.

Building an IPS system requires solving two problems. *First*, the search problem: to find consistent programs we need to search over a suitable set of possible programs. We need to define the set (i.e., the program space) and search procedure. *Second*, the ranking problem: if there are multiple programs consistent with the input-output examples, which one do we return? Both of these problems are dependent on the specifics of the problem formulation. Thus, the first important decision in formulating an approach to program synthesis is the choice of a *Domain Specific Language*.

Domain Specific Languages (DSLs). DSLs are programming languages that are suitable for a specialized domain but are more restrictive than full-featured programming languages. For example, one might disallow loops or other control flow, and only allow string data types and a small number of primitive operations like concatenation. Most of program synthesis research focuses on synthesizing programs in DSLs, because full-featured languages like C++ enlarge the search space and complicate synthesis. Restricted DSLs can also enable more efficient special-purpose search algorithms. For example, if a DSL only allows concatenations of substrings of an input string, a dynamic programming algorithm can efficiently search over all possible programs (Polozov and Gulwani, 2015). The choice of DSL also affects the difficulty of the ranking problem.

For example, in a DSL without `if` statements, the same algorithm is applied to all inputs, reducing the number of programs consistent with any set of input-output examples, and thus the ranking problem becomes easier. Of course, the restrictiveness of the chosen DSL also determines which problems the system can solve at all.

Search Techniques. There are many techniques for searching for programs consistent with input-output examples. Perhaps the simplest approach is to define a grammar and then enumerate all derivations of the grammar, checking each one for consistency with the examples. This approach can be combined with pruning based on types and other logical reasoning (Feser et al., 2015). While simple, these approaches can be implemented efficiently, and they can be surprisingly effective.

In restricted domains such as the concatenation example discussed above, special-purpose algorithms can be used. FlashMeta (Polozov and Gulwani, 2015) describes a framework for DSLs which allow decomposition of the search problem, e.g., where the production of an output string from an input string can be reduced to finding a program for producing the first part of the output and concatenating it with a program for producing the latter part of the output string.

Another class of systems is based on Satisfiability Modulo Theories (SMT) solving. SMT combines SAT-style search with *theories* like arithmetic and inequalities, with the benefit that theory-dependent subproblems can be handled by special-purpose solvers. For example, a special-purpose solver can easily find integers x, y such that $x < y$ and $y < -100$ hold, whereas an enumeration strategy may need to consider many values before satisfying the constraints. Many program synthesis engines based on SMT solvers exist, e.g., Sketch (Solar-Lezama, 2008) and Brahma (Gulwani et al., 2011). They convert the semantics of a DSL into a set of constraints between variables representing the program and the input-output values, and then call an SMT solver to find a satisfying setting of the program variables. This approach shines when special-purpose reasoning can be leveraged, but complex DSLs can lead to very large constraint problems where constructing and manipulating the constraints can be a lot slower than an enumerative approach.

Finally, stochastic local search can be employed to search over program space, and there is a long history of applying genetic algorithms to this problem. One of the most successful recent examples is the STOKE super-optimization system (Schkufza et al., 2016), which uses stochastic local search to find assembly programs that have the same semantics as an input program but execute faster.

Ranking. While we focus on the search problem in this chapter, we briefly mention the ranking problem here. A popular choice for ranking is to choose the shortest program consistent with input-output examples (Gulwani, 2016). A more sophisticated approach is employed by FlashFill (Singh and Gulwani, 2015). It works in a manner similar to max-margin structured prediction, where known ground truth programs are given, and the learning task is to assign scores to programs such that the ground truth programs score higher than other programs that satisfy the input-output specification.

5.3 Learning Inductive Program Synthesis (LIPS)

In this section we outline the general approach that we follow in this chapter, which we call *Learning Inductive Program Synthesis* (LIPS). The details of our instantiation of LIPS appear in Section 5.4. The components of LIPS are (1) a DSL specification, (2) a data-generation procedure, (3) a machine learning model that maps from input-output examples to program attributes, and (4) a search procedure that searches program space in an order guided by the model from (3). The framework is related to the formulation of Menon et al. (2013); the relationship and key differences are discussed in Section 5.6.

(1) DSL and Attributes. The choice of DSL is important in LIPS, just as it is in any program synthesis system. It should be expressive enough to capture the problems that we wish to solve, but restricted as much as possible to limit the difficulty of the search. In LIPS we additionally specify an *attribute function* $\mathcal{A}$ that maps programs P of the DSL to finite *attribute vectors* $\boldsymbol{a} = \mathcal{A}(P)$. (Attribute vectors of different programs need not have equal length.) Attributes serve as the link between the machine learning and the search component of LIPS: the machine learning model predicts a distribution $q(\boldsymbol{a} \mid \mathcal{E})$, where $\mathcal{E}$ is the set of input-output examples, and the search procedure aims to search over programs P as ordered by $q(\mathcal{A}(P) \mid \mathcal{E})$. Thus an attribute is useful if it is both predictable from input-output examples, and if conditioning on its value significantly reduces the effective size of the search space.

Possible attributes are the (perhaps position-dependent) presence or absence of high-level functions (e.g., does the program contain or end in a call to Sort). Other possible attributes include control flow templates (e.g., the number of loops and conditionals). In the extreme case, one may set $\mathcal{A}$ to the identity function, in which case the attribute

is equivalent to the program; however, in our experiments we find that performance is improved by choosing a more abstract attribute function.

(2) Data Generation. Step 2 is to generate a dataset $((P^{(n)}, \boldsymbol{a}^{(n)}, \mathcal{E}^{(n)}))_{n=1}^{N}$ of programs $P^{(n)}$ in the chosen DSL, their attributes $\boldsymbol{a}^{(n)}$, and accompanying input-output examples $\mathcal{E}^{(n)}$. Different approaches are possible, ranging from enumerating valid programs in the DSL and pruning, to training a more sophisticated generative model of programs in the DSL. The key in the LIPS formulation is to ensure that it is feasible to generate a large dataset (ideally millions of programs).

(3) Machine Learning Model. The machine learning problem is to learn a distribution of attributes given input-output examples, $q(\boldsymbol{a} \mid \mathcal{E})$. There is freedom to explore a large space of models, so long as the input component can encode $\mathcal{E}$, and the output is a proper distribution over attributes (e.g., if attributes are a fixed-size binary vector, then a neural network with independent sigmoid outputs is appropriate; if attributes are variable size, then a recurrent neural network output could be used). Attributes are observed at training time, so training can use a maximum likelihood objective.

(4) Search. The aim of the search component is to interface with an existing solver, using the predicted $q(\boldsymbol{a} \mid \mathcal{E})$ to guide the search. We describe specific approaches in the next section.

5.4 DeepCoder

Here we describe DeepCoder, our instantiation of LIPS including a choice of DSL, a data generation strategy, models for encoding input-output sets, and algorithms for searching over program space.

5.4.1 Domain Specific Language and Attributes

We consider binary attributes indicating the presence or absence of high-level functions in the target program. To make this effective, the chosen DSL needs to contain constructs that are not so low-level that they all appear in the vast majority of programs, but at the same time should be common enough so that predicting their occurrence from input-output examples can be learned successfully.

Following this observation, our DSL is loosely inspired by query languages such as SQL or LINQ, where high-level functions are used in sequence to manipulate data. A program in our DSL is a sequence of function calls, where the result of each call initializes a fresh variable that is either a singleton integer or an integer array. Functions can be applied to any of the inputs or previously computed (intermediate) variables. The output of the program is the return value of the last function call, i.e., the last variable. See Figure 5.1 for an example program of length $T = 4$ in our DSL.

`a` ← `[int]`	**An input-output example:**
`b` ← FILTER `(<0) a`	*Input*:
`c` ← MAP `(*4) b`	`[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]`
`d` ← SORT `c`	*Output*:
`e` ← REVERSE `d`	`[-12, -20, -32, -36, -68]`

Fig. 5.1 An example program in our DSL that takes a single integer array as its input.

Overall, our DSL contains the first-order functions HEAD, LAST, TAKE, DROP, ACCESS, MINIMUM, MAXIMUM, REVERSE, SORT, SUM, and the higher-order functions MAP, FILTER, COUNT, ZIPWITH, SCANL1. Higher-order functions require suitable lambda functions for their behavior to be fully specified: for MAP our DSL provides lambdas `(+1)`, `(-1)`, `(*2)`, `(/2)`, `(*(-1))`, `(**2)`, `(*3)`, `(/3)`, `(*4)`, `(/4)`; for FILTER and COUNT there are predicates `(>0)`, `(<0)`, `(%2==0)`, `(%2==1)` and for ZIPWITH and SCANL1 the DSL provides lambdas `(+)`, `(-)`, `(*)`, MIN, MAX. A description of the semantics of all functions is provided in Appendix D.6.

Note that while the language only allows linear control flow, many of its functions do perform branching and looping internally (e.g., SORT, COUNT, ...). Examples of more sophisticated programs expressible in our DSL, which were inspired by the simplest problems appearing on programming competition websites, are shown in Appendix D.1.

5.4.2 Data Generation

To generate a dataset, we enumerate programs in the DSL, heuristically pruning away those with easily detectable issues such as a redundant variable whose value does not affect the program output, or, more generally, existence of a shorter equivalent program (equivalence can be overapproximated by identical behavior on randomly or carefully chosen inputs). To generate valid inputs for a program, we enforce a constraint on the output value bounding integers to some predetermined range, and then propagate these constraints backward through the program to obtain a range of valid values for each

input. If one of these ranges is empty, we discard the program. Otherwise, input-output pairs can be generated by picking inputs from the pre-computed valid ranges and executing the program to obtain the output values. The binary attribute vectors are easily computed from the program source codes.

5.4.3 Machine Learning Model

Observe how the input-output data in Figure 5.1 is informative of the functions appearing in the program: the values in the output are all negative, divisible by 4, they are sorted in decreasing order, and they happen to be multiples of numbers appearing in the input. Our aim is to learn to recognize such patterns in the input-output examples, and to leverage them to predict the presence or absence of individual functions. We employ neural networks to model and learn the mapping from input-output examples to attributes. We can think of these networks as consisting of two parts:

1. an *encoder*: a differentiable mapping from a set of M input-output examples generated by a single program to a latent real-valued vector, and
2. a *decoder*: a differentiable mapping from the latent vector representing a set of M input-output examples to predictions of the ground truth program's attributes.

For the encoder we use a simple feed-forward architecture. First, we represent the input and output types (singleton or array) by a one-hot-encoding, and we pad the inputs and outputs to a maximum length L with a special NULL value. Second, each integer in the inputs and in the output is mapped to a learned embedding vector of size $E = 20$. (The range of integers is restricted to a finite range and each embedding is parametrized individually.) Third, for each input-output example separately, we concatenate the embeddings of the input types, the inputs, the output type, and the output into a single (fixed-length) vector, and pass this vector through $H = 3$ hidden layers containing $K = 256$ sigmoid units each. The third hidden layer thus provides an encoding of each individual input-output example. Finally, for input-output examples in a set generated from the same program, we pool these representations together by simple arithmetic averaging. See Appendix D.3 for more details on this architecture.

The advantage of this encoder lies in its simplicity, and we found it reasonably easy to train. A disadvantage is that it requires an upper bound L on the length of arrays appearing in the input and output. We confirmed that the chosen encoder architecture is sensible in that it performs empirically at least as well as an RNN encoder, a natural baseline, which may however be more difficult to train.

DeepCoder learns to predict presence or absence of individual functions of the DSL. We shall see this can already be exploited by various search techniques to large computational gains. We use a decoder that pre-multiplies the encoding of input-output examples by a learned $C \times K$ matrix, where $C = 34$ is the number of functions in our DSL (higher-order functions and lambdas are predicted independently), and treats the resulting C numbers as log-unnormalized probabilities (logits) of each function appearing in the source code. Figure 5.2 shows the predictions a trained neural network made from 5 input-output examples for the program shown in Figure 5.1.

Fig. 5.2 Neural network predicts the probability of each function appearing in the source code. All instructions appearing in the ground truth program, shown in Figure 5.1, are correctly identified in this case.

5.4.4 Search

One of the central ideas of this work is to use a neural network to guide the search for a program consistent with a set of input-output examples instead of directly predicting the entire source code. This section briefly describes the search techniques and how they integrate the predicted attributes.

Depth-first search (DFS). We use an optimized version of DFS to search over programs with a given maximum length T (see Appendix D.4 for details). When the search procedure extends a partial program by a new function, it has to try the functions in the DSL in some order. At this point DFS can opt to consider the functions as ordered by their predicted probabilities from the neural network.

"Sort and add" enumeration. A stronger way of utilizing the predicted probabilities of functions in an enumerative search procedure is to use a *Sort and add* scheme, which maintains a set of *active* functions and performs DFS with the active function set only. Whenever the search fails, the next most probable function (or several) are added to the active set and the search restarts with this larger active set. Note that this scheme has the deficiency of potentially re-exploring some parts of the search space several times, which could be avoided by a more sophisticated search procedure.

Sketch. Sketch (Solar-Lezama, 2008) is a successful SMT-based program synthesis tool from the programming languages research community. While its main use case is to synthesize programs by filling in "holes" in incomplete source code so as to match specified requirements, it is flexible enough for our use case as well. The function in each step and its arguments can be treated as the "holes", and the requirement to be satisfied is consistency with the provided set of input-output examples. Sketch can utilize the neural network predictions in a *Sort and add* scheme as described above, as the possibilities for each function hole can be restricted to the current active set.

λ^2. λ^2 (Feser et al., 2015) is a program synthesis tool from the programming languages community that combines enumerative search with deduction to prune the search space. It is designed to infer small functional programs for data structure manipulation from input-output examples, by combining functions from a provided library. λ^2 can be used in our framework using a *Sort and add* scheme as described above by choosing the library of functions according to the neural network predictions.

5.4.5 Training Loss Function

We use the negative cross entropy loss to train the neural network described in Section 5.4.3, so that its predictions about each function can be interpreted as marginal probabilities. The LIPS framework dictates learning $q(\boldsymbol{a} \mid \mathcal{E})$, the joint distribution of all attributes $\boldsymbol{a}$ given the input-output examples, and it is not clear a priori how much DeepCoder loses by ignoring correlations between functions. However, under the simplifying assumption that the runtime of searching for a program of length T with C functions made available to a search routine is proportional to C^T, the following result for *Sort and add* procedures shows that their runtime can be optimized using only the marginal probabilities.

Lemma 2. *For any fixed program length T, the expected total runtime of a* Sort and add *search scheme can be upper bounded by a quantity that is minimized by adding the functions in the order of decreasing true marginal probabilities.*

Proof. Predicting source code functions from input-output examples can be seen as a multi-label classification problem, where each set of input-output examples is associated with a set of relevant labels (functions appearing in the ground truth source code). Dembczynski et al. (2010) showed that in multi-label classification under a so-called *Rank loss* (number of pairs of labels that need to be swapped before positive labels are

all ranked higher than negative labels), it is Bayes optimal to rank the labels according to their marginal probabilities. If the runtime of search with C functions is proportional to C^T, the total runtime of a *Sort and add* procedure can be monotonically transformed so that it is upper bounded by this Rank loss. See Appendix D.5 for more details. □

5.5 Experiments

In this section we report results from two categories of experiments. Our main experiments (Section 5.5.1) show that the LIPS framework can lead to significant performance gains in solving IPS by demonstrating such gains with DeepCoder. In Section 5.5.2 we illustrate the robustness of the method by demonstrating a strong kind of generalization ability across programs of different lengths.

5.5.1 DeepCoder Compared to Baselines

We trained a neural network as described in Section 5.4.3 to predict used functions from input-output examples and constructed a test set of $P = 500$ programs, guaranteed to be semantically disjoint from all programs on which the neural network was trained (similarly to the equivalence check described in Section 5.4.2, we have ensured that all test programs behave differently from all programs used during training on at least one input). For each test program we generated $M = 5$ input-output examples involving integers of magnitudes up to 256, passed the examples to the trained neural network, and fed the obtained predictions to the search procedures from Section 5.4.4. We also considered a RNN-based decoder generating programs token-by-token using beam search (see Section 5.5.3 for details). To evaluate DeepCoder, we then recorded the time the search procedures needed to find a program consistent with the M input-output examples. As a baseline, we also ran all search procedures using a simple prior as function probabilities, computed from their global incidence in the program corpus.

Table 5.1 Search speedups on programs of length $T = 3$ due to using neural network predictions.

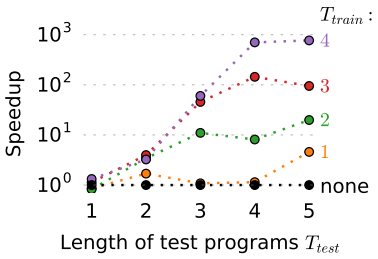
Timeout needed	**DFS**			**Sort and add**			**λ^2**			**Sketch**		**Beam**
to solve	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	20%
Baseline	41ms	126ms	314ms	80ms	335ms	861ms	18.9s	49.6s	84.2s	$>10^3s$	$>10^3s$	$>10^3s$
DeepCoder	2.7ms	33ms	110ms	1.3ms	6.1ms	27ms	0.23s	0.52s	13.5s	2.13s	455s	292s
Speedup	**15.2×**	**3.9×**	**2.9×**	**62.2×**	**54.6×**	**31.5×**	**80.4×**	**94.6×**	**6.2×**	**>467×**	**>2.2×**	**>3.4×**

In the first, smaller-scale experiment (program search space size $\sim 2 \times 10^6$) we trained the neural network on programs of length $T = 3$, and the test programs were of the same length. Table 5.1 shows the per-task timeout required such that a solution could be found for given proportions of the test tasks (in time less than or equal to the timeout). For example, in a hypothetical test set with 4 tasks and runtimes of $3s$, $2s$, $1s$, $4s$, the timeout required to solve 50% of tasks would be $2s$. More detailed experimental results are discussed in Appendix D.2.

In the main experiment, we tackled a large-scale problem of searching for programs consistent with input-output examples generated from programs of length $T = 5$ (search space size on the order of 10^{10}), supported by a neural network trained with programs of shorter length $T = 4$. Here, we only consider $P = 100$ programs for reasons of computational efficiency, after having verified that this does not significantly affect the results in Table 5.1. The table in Figure 5.3a shows significant speedups for DFS, *Sort and add* enumeration, and λ^2 with *Sort and add* enumeration, the search techniques capable of solving the search problem in reasonable time frames. Note that *Sort and add* enumeration without the neural network (using prior probabilities of functions) exceeded the 10^4 second timeout in two cases, so the relative speedups shown are crude lower bounds.

Timeout needed	**DFS**			**Sort and add**			λ^2
to solve	20%	40%	60%	20%	40%	60%	20%
Baseline	$163s$	$2887s$	$6832s$	$8181s$	$>10^4s$	$>10^4s$	$463s$
DeepCoder	$24s$	$514s$	$2654s$	$9s$	$264s$	$4640s$	$48s$
Speedup	**6.8×**	**5.6×**	**2.6×**	**907×**	**>37×**	**>2×**	**9.6×**

(a)



(b)

Fig. 5.3 (a) Search speedups on test programs of length $T = 5$. (b) Influence of length of training programs on the speedup.

We hypothesize that the substantially larger performance gains on *Sort and add* schemes as compared to gains on DFS can be explained by the fact that the choice of attribute function (predicting presence of functions anywhere in the program) and learning objective of the neural network are better matched to the *Sort and add* schemes. Indeed, a more appropriate attribute function for DFS would be one that is more informative of the functions appearing early in the program, since exploring an incorrect first function is costly with DFS. On the other hand, the discussion in Section 5.4.5 provides theoretical indication that ignoring the correlations between functions is not

cataclysmic for *Sort and add* enumeration, since a Rank loss that upper bounds the *Sort and add* runtime can still be minimized.

In Appendix D.7 we analyse the performance of the neural networks used in these experiments, by investigating which attributes (program instructions) tend to be difficult to distinguish from each other.

5.5.2 Generalization Across Program Lengths

To investigate the neural network's generalization ability across programs of different lengths, we trained a network to predict used functions from input-output examples that were generated from programs of length $T_{\text{train}} \in \{1, \dots, 4\}$. We then used each of these networks to predict functions on 5 test sets containing input-output examples generated from programs of lengths $T_{\text{test}} \in \{1, \dots, 5\}$, respectively. The test programs of a given length T were semantically disjoint from all training programs of the same length T and also from all training and test programs of shorter lengths $T' < T$.

For each of the combinations of T_{train} and T_{test}, *Sort and add* enumerative search was run both with and without using the neural network's predictions (in the latter case using prior probabilities) until it solved 20% of the test set tasks. Figure 5.3b shows the relative speedup of the solver having access to predictions from the trained neural networks. These results indicate that the neural networks are able to generalize beyond programs of the same length that they were trained on. This is partly due to the search procedure on top of their predictions, which has the opportunity to correct for the presence of functions that the neural network failed to predict. Note that a sequence-to-sequence model trained on programs of a fixed length would not ordinarily be expected to exhibit this kind of generalization ability.

5.5.3 Alternative Models

Encoder We evaluated replacing the feed-forward architecture encoder (Section 5.4.3) with an RNN, a natural baseline. Using a GRU-based RNN we were able to achieve results almost as good as using the feed-forward architecture, but found the RNN encoder more difficult to train.

Decoder We also considered a purely neural network-based approach, where an RNN decoder is trained to predict the entire program token-by-token. We combined this with our feed-forward encoder by initializing the RNN using the pooled final layer

of the encoder. We found it substantially more difficult to train an RNN decoder as compared to the independent binary classifiers employed above. Beam search was used to explore likely programs predicted by the RNN, but it only lead to a solution comparable with the other techniques when searching for programs of lengths $T \leq 2$, where the search space size is very small (on the order of 10^3). Note that using an RNN for both the encoder and decoder corresponds to a standard sequence-to-sequence model. However, we do do not rule out that a more sophisticated RNN decoder or training procedure could be possibly more successful.

5.6 Related Prior Work

Machine Learning for Inductive Program Synthesis. There is relatively little prior work on using machine learning for programming by example. The most closely related work is that of Menon et al. (2013), in which a hand-coded set of features of input-output examples are used as "clues." When a clue appears in the input-output examples (e.g., the output is a permutation of the input), it reweights the probabilities of productions in a probabilistic context free grammar by a learned amount. This work shares the idea of learning to guide the search over program space conditional on input-output examples. One difference is in the domains. Menon et al. (2013) operate on short string manipulation programs, where it is arguably easier to hand-code features to recognize patterns in the input-output examples (e.g., if the outputs are always permutations or substrings of the input). Our work shows that there are strong cues in patterns in input-output examples in the domain of numbers and lists. However, the main difference is the scale. Menon et al. (2013) learns from a small (280 examples), manually-constructed dataset, which limits the capacity of the machine learning model that can be trained. Thus, it forces the machine learning component to be relatively simple. Indeed, Menon et al. (2013) use a log-linear model and rely on hand-constructed features. LIPS automatically generates training data, which yields datasets with millions of programs and enables high-capacity deep learning models to be brought to bear on the problem.

Learning Representations of Program State. Piech et al. (2015) propose to learn joint embeddings of program states and programs to automatically extend teacher feedback to many similar programs in the MOOC setting. This work is similar in that it considers embedding program states, but the domain is different, and it otherwise

specifically focuses on syntactic differences between semantically equivalent programs to provide stylistic feedback. Li et al. (2016) use graph neural networks (GNNs) to predict logical descriptions from program states, focusing on data structure shapes instead of numerical and list data. Such GNNs may be a suitable architecture to encode states appearing when extending our DSL to handle more complex data structures.

Learning to Infer. Alemi et al. (2016) used neural sequence models in tandem with an automated theorem prover: similar to our *Sort and Add* strategy, a neural network component is trained to select premises that the theorem prover can use to prove a theorem. A follow-up extension (Loos et al., 2017) is similar to our DFS enumeration strategy and uses a neural network to guide the proof search at intermediate steps. By reducing the search space like this, more theorems can be proven in the given time limits. The main differences from DeepCoder are in the domains, and that they train on an existing corpus of theorems. More broadly, if we view a DSL as defining a model and search as a form of inference algorithm, then there is a large body of work on using discriminatively-trained models to aid inference in generative models. Examples include (Dayan et al., 1995; Heess et al., 2013; Jampani et al., 2015; Kingma and Welling, 2014; Le et al., 2017; Shotton et al., 2013; Stuhlmüller et al., 2013).

5.7 Discussion

We have presented a framework for improving program synthesis systems by using neural networks to translate cues in input-output examples to guidance over where to search in program space. Our empirical results show that for many programs, this technique improves the runtime of a wide range of synthesis baselines by 1-3 orders of magnitude. We have found several problems in real online programming competitions that can be solved with a program in our language, which validates the relevance of the class of problems that we have studied in this work. In sum, this suggests that we have made good progress towards being able to solve programming competition problems, and the machine learning component plays an important role in making it tractable.

Limitations There remain some immediate limitations, however. First, the programs we can synthesize are only the simplest problems on programming competition websites and are simpler than most competition problems. Many problems require more complex algorithmic solutions like dynamic programming and search, which are currently beyond

our reach. Our chosen DSL currently cannot express solutions to many problems. To do so, it would need to be extended by adding more primitives and allow for more flexibility in program constructs (such as allowing loops). Second, we currently use five input-output examples with relatively large integer values (up to 256 in magnitude), which are probably more informative than typical (smaller) examples. While we remain optimistic about the applicability of our approach as the DSL becomes more complex and the input-output examples become less informative, it remains to be seen what the magnitude of these effects are as we move towards solving large subsets of programming competition problems.

5.7.1 Future Directions

Apart from the more immediate limitations of applying DeepCoder to automatically solving programming competition style problems listed above, there are multiple conceptual differences between the presented systems and, say, a human attempting to solve such a task by themselves. Each of these conceptual points forms an interesting direction of future research. In fact, some have already been undertaken since first publication of DeepCoder (Balog et al., 2017a), and some of that work is also highlighted below.

- Crude **global properties**
  This is a non-fundamental limitation of DeepCoder, where for reasons of simplicity only the presence or absence of each DSL primitive anywhere in the program was predicted. This is clearly sub-optimal as this approach (1) quickly fails for longer programs, and (2) does not account for the fact that there could exist alternative solutions to the same problem that use very different instruction sets. *RobustFill* (Devlin et al., 2017) addressed this by training a neural network directly outputting programs token-by-token, an approach that we did not get to work that well in the DeepCoder domain. *PCCoder* (Zohar and Wolf, 2018) and Ellis et al. (2019) addressed this by setting up the problem in a way that only the first line of the program needs to be predicted at any time. Somewhere in between lies *SketchAdapt* (Nye et al., 2019), where a neural network is trained to output a program sketch that can contain holes in arbitrary places, as decided by the network.

- Problem **decomposition** (hierarchical reasoning)
  DeepCoder does not have a notion of coming up with a general strategy for

approaching a given problem, and only then filling in the details. Given a set of input-output examples DeepCoder makes one set of predictions about which instructions should be used. It does not have the opportunity to make a refined set of predictions after some options have been tried, or parts of the code have been filled in. What can be viewed as a step towards addressing this limitation, *SketchAdapt* (Nye et al., 2019) employed a two stage procedure where first a program sketch is predicted by a neural network, but holes can be left in places where the network is not confident enough. In the second stage, a DeepCoder-like system is utilized to make predictions useful for filling in the remaining holes. However, the first stage is not quite a problem decomposition, because the generated sketch does not come with a specification what each hole should (roughly) do.

Another avenue towards addressing this problem could go via execution traces, which could be predicted for each given input-output example, then aligned and thus decomposition points could be identified, with each component then solved recursively.

- Ability to **execute**
  Although the Sketch search algorithm did know about the semantics of individual DSL operations and could use them in its search to find a program matching the given input-output examples, the search algorithm that worked best for us (DFS) does not have the ability to execute – programs are only executed "outside the search" after all lines of the program have been filled in and it is to be checked whether the prediction matches the given input-output examples (or whether the search needs to backtrack). *PCCoder* (Zohar and Wolf, 2018) and (Ellis et al., 2019) directly addressed this conceptual limitation by training the network to predict one line of code at a time, and then executing it on the current program state. The resulting program state is then fed as input to the network for predicting the following line. This is iterated until the program state obtained after executing the most recently generated line matches the desired output.

- Explicit **language model**
  Experienced human programmers have a notion of how sensible computer programs for a given task should look like. As a trivial example, defining unused variables or performing irrelevant actions are clear indications that the program

is at the very least written in an unexpected way, but more likely that there is a bug in the source code. While a neural network making program predictions can implicitly learn language models of source code, *SketchAdapt* (Nye et al., 2019) found it helpful to add an LSTM language model of source code on top of the sketch generator, in order to reweight the sequences output by that component.

- Realism of **training data**
  Even if the neural network predicting source code is capable of learning a good language model of source code, it will only do so if it is trained on realistic data. DeepCoder and most of follow-up work have primarily used synthetic data for training, where training programs are sampled uniformly at random from a specific DSL. Moreover, input-output examples are also generated by uniform random sampling, at best with added heuristics or rejection sampling to ensure that programs do not crash on their generated inputs. Shin et al. (2019b) highlighted issues arising from the mismatch between synthetic training distributions and real-world distributions of programs and inputs, and together with Clymo et al. (2019) proposed first solution steps based on the idea of encouraging diversity in the generated synthetic data.

- Growing set of reusable **building blocks**
  In DeepCoder and most other approaches the DSL is fixed upfront, and the network can only implicitly recognize that some sequence of operations is frequently useful as a unit; and even if it does, the network cannot predict such a unit in a single step. Shin et al. (2019a) identified this limitation and addressed it by mining common code idioms from a given corpus of code and showed that it can aid program synthesis in a semantic parsing setup.

- Interaction between **pattern matching** and **symbolic reasoning**
  DeepCoder performs a single step of pattern matching on a fixed level of granularity (presence or absence of DSL primitives), and then offloads the predictions to symbolic reasoning (a search algorithm). *SketchAdapt* (Nye et al., 2019) improves upon this by automatically choosing the level at which pattern matching stops and symbolic reasoning takes over by allowing the initial sketch generator to leave arbitrarily large holes. Switching back and forth between symbolic reasoning and pattern matching is still not explicitly supported, although execution-guided program synthesis (Chen et al., 2019; Zohar and Wolf, 2018) and mining code

idioms at different levels of granularity (Shin et al., 2019b) can be seen as steps in that direction.

- **Early feedback**
  A human programmer can sometimes realize during the process of writing source code that the chosen approach is unlikely to work. This realization might come from syntax (the code looks unnatural), or from semantics (partial execution of parts of the code). In such a situation the programmer would backtrack, but DeepCoder does not have a similar mechanism, at least not explicitly (the Sketch synthetizer may prune some parts of the search space due to semantic constraints). Ellis et al. (2019) introduced a learned *value function* into the synthesis search loop, so that unpromising directions can be weighted down early on. While any program generating (policy) network could in principle learn to only generate promising directions, training an explicit value network has been shown to be helpful.

- Generating **new test cases**
  For a human programmer it can be very instructive to see the behaviour (execution) of a (partial or complete) program on new test cases, perhaps ones covering corner cases or ones that are even simpler than in the originally provided input-output examples. Laich et al. (2020) proposed a system that generates new inputs for the program being synthesized, with a trained neural network oracle predicting whether a potential output is desirable or not. However, the focus of this work is on the ranking problem (deciding which consistent program is the one desired by the user) rather than on the search problem.

- Utilizing **side information**
  Faced with the problem of finding a program that solves a particular task, it is clearly beneficial to take into account as much information about the task as possible. DeepCoder has focused on utilizing an input-output example specification, but bringing in other side information such as the context of the task (perhaps the type of programming contest) and in particular a natural language problem statement should help. For a DeepCoder-style approach to apply, the challenge is to obtain sufficient amounts of training data that would include task descriptions in natural language. Such datasets have been created for and evaluated using bash (Lin et al., 2017), SQL (Zhong et al., 2017) and a DSL that can solve a particular type of computer science homework assignments (Polosukhin

and Skidanov, 2018). The latter has been used to evaluate the *SketchAdapt* (Nye et al., 2019) method that had already been mentioned above.

Chapter 6

Conclusion

The central philosophical idea of this thesis is that most computational problems arising in machine learning are on a mathematical level instances of one of the following four abstract tasks: *integration*, *sampling*, *optimization*, and *search*. While exceptions exist (as discussed in Section 1.2.5), Sections 1.2.1-1.2.4 corroborated this idea by providing an overview of a wide range of machine learning problems in each of the four categories. Section 1.3 then introduced the concept of *conversions*, which are efficient computations that transform a problem of one abstract type into a problem of another abstract type, such that solving the latter problem yields or helps finding a solution to the original problem. Multiple reasons for why a conversion can be preferable to solving the original problem were also discussed in Section 1.3. Section 1.4 provided an overview of existing machine learning algorithms that can be seen as performing such conversions, with a special focus on conversions to optimization problems (Section 1.4.1) as the novel methods presented in this thesis also convert to optimization problems.

The main contributions of this thesis were four new algorithms, or generalizations of existing algorithms, that perform a conversion from a non-optimization problem type to an optimization problem.

Chapter 2 generalized the Gumbel trick, a method for converting the *integration* problem of partition function estimation into the *optimization* problem of finding a most likely configuration. The generalization led to an entire family of tricks, of which the Gumbel trick is just one member. Other members of the family were shown to be statistically more efficient in various application domains of the Gumbel trick, including low-rank perturbation approximations used in discrete graphical models. Using one of the newly derived methods one can construct a statistical estimator of the partition

function that achieves the same mean squared error while using (up to 40%) fewer samples, where obtaining each sample entails solving an MAP optimization problem.

Chapter 3 generalized the Gumbel trick in an orthogonal direction, showing how a different *integration* problem of estimating Rényi entropies can also be converted into the *optimization* problem of finding a most likely configuration. The proposed algorithm yields unbiased estimators of the Rényi entropies, and thanks to its simplicity applies whenever the original Gumbel trick does. Chapters 2 and 3 were both instances of conversions where the conversion is beneficial because it maps onto a well studied problem of finding a maximum probability configuration in a discrete graphical model.

Chapter 4 proposed a conversion from the *sampling* problem dictated by a differential privacy requirement when wishing to release a sensitive database to the public, to an *optimization* problem in a Reproducing Kernel Hilbert Space. In the context of this thesis, the main benefit of this conversion is that it allows traversing the computation-accuracy-privacy trade-off manifold in a qualitatively different way. Approximating the sampling problem without conversion to optimization would usually trade off all three resources (computation, accuracy, privacy) at the same time, with the privacy loss incurred being hard to quantify. The proposed method allows trading off accuracy with computation while always respecting a fixed privacy budget. Moreover, the mathematical theory of Reproducing Kernel Hilbert Spaces allowed proving consistency results and finite-sample convergence bounds for the estimators that are released to the public.

Chapter 5 showed how the *discrete search* problem of finding a program source code consistent with a given input-output specification can be approached by solving a *continuous optimization* problem. By casting program synthesis as a supervised learning problem the original difficult search problem could be attacked using effective gradient-based methods for training deep neural networks. The trained neural network could potentially be used to directly solve discrete search problem instances (we considered such an end-to-end RNN baseline in Section 5.5.3), thus providing a direct conversion from discrete search to continuous optimization. Interestingly however, it was beneficial to combine the strengths of the methods developed for the original problem (synthesizers from the programming languages community) with the pattern-matching abilities of the neural networks applied to the problem after conversion. This highlights that conversions between different abstract problem types can also be exploited to synergize strengths of methods developed by different (sub)fields of computer science.

Future work Each of the four main contributions in this thesis has opened up multiple avenues for future work, which have been described in detail in the respective Sections 2.6, 3.5, 4.7, and 5.7. Here we describe directions related to the overarching topic of the thesis, converting between different abstract problem types.

Different generalizations of the Gumbel trick presented in Chapters 2 and 3 can be more efficient in different settings, and a better analytical understanding of which conversion can be expected to work best for a given problem instance would be desirable. Looking forward, the conversion to optimization problems using the developed techniques could be also feasible for other integration problems beyond partition function estimation and Rényi entropy estimation (see Section 3.5).

As shown by Chapter 4, performing a conversion can unlock new trade-offs between different resource types. It will be interesting to investigate whether conversions can be also useful in other domains where multiple kinds of resources (computation time, memory, privacy, fairness, statistical efficiency, randomness,...) need to be balanced.

The empirical results of Chapter 5 demonstrated that conversions can be utilized not only to replace one abstract problem type with another, but also to combine the strengths of methods available for both the original and the new problem type. This is a general observation that could be applicable in many problems where conversions are available.

Finally, on a high level, this thesis has treated the concept of conversions informally. A formalization of the concept would be desirable to gain a deeper understanding of when conversions can be useful, and how they can trade off different resource types.

Final words With basic research it is often difficult to predict its impact ahead of time. The work presented in this thesis is hopefully well-positioned to be useful, as the tasks considered are fairly general and address important issues arising in machine learning:

- The task of estimating normalizing constants (partition function estimation) and the task of estimating uncertainty in a distribution (entropy estimation) considered in Chapters 2 and 3, respectively, are ubiquitous in (probabilistic) machine learning. Since probabilistic reasoning is a fundamental concept when interacting with the real world, these tasks are unlikely to go away, and they may also show up in new settings where converting to optimization can again be useful.

- With the rise of statistical and machine learning techniques the issue of protecting individual privacy has been attracting increasing amounts of attention not only within the scientific community, but also in the general public, in the media, and in politics. With a lack of clear guidelines there is substantial risk of both under-regulation and over-regulation of the use of statistical techniques on valuable data, such as patient health records. The theoretical work of Chapter 4 on safe methods for releasing sensitive data so that useful statistical inferences can be drawn from it without violating individual privacy is therefore a step towards addressing one of the challenges of our time.

- Finally, the work on program synthesis from Chapter 5 has already been picked up and expanded on by the research community aiming to build useful synthesis systems. While these systems may well turn out to be practically useful for software engineers or even non-experts already in the short term, it is also conceivable that program synthesis techniques will form a crucial stepping stone towards building robust autonomous agents in the quest towards artificial intelligence.

With these grandiose goals stated, it is important to emphasize that research in general and the works presented in this thesis in particular usually proceed through small steps. And many more such steps will be needed.

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *ACM SIGSAC Conference on Computer and Communications Security*.

Acharya, J., Orlitsky, A., Suresh, A. T., and Tyagi, H. (2016). Estimating Rényi entropy of discrete distributions. *IEEE Transactions on Information Theory*.

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive science*.

Aldà, F. and Rubinstein, B. I. P. (2017). The Bernstein mechanism: Function release under differential privacy. In *AAAI Conference on Artificial Intelligence*.

Alemi, A. A., Chollet, F., Irving, G., Szegedy, C., and Urban, J. (2016). DeepMath - deep sequence models for premise selection. In *Advances in Neural Information Processing Systems*.

Allamanis, M., Brockschmidt, M., and Khademi, M. (2018). Learning to represent programs with graphs. In *International Conference on Learning Representations*.

Alon, N., Matias, Y., and Szegedy, M. (1999). The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*.

Baez, J. C. (2011). Rényi entropy and free energy. *arXiv:1102.2098*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., and Tarlow, D. (2017a). DeepCoder: Learning to write programs. In *International Conference on Learning Representations*.

Balog, M., Lakshminarayanan, B., Ghahramani, Z., Roy, D. M., and Teh, Y. W. (2016). The Mondrian kernel. In *Uncertainty in Artificial Intelligence*.

Balog, M., Tolstikhin, I., and Schölkopf, B. (2018). Differentially private database release via kernel mean embeddings. In *International Conference on Machine Learning*.

Balog, M., Tripuraneni, N., Ghahramani, Z., and Weller, A. (2017b). Lost relatives of the Gumbel trick. In *International Conference on Machine Learning*.

Bavishi, R., Lemieux, C., Fox, R., Sen, K., and Stoica, I. (2019). Autopandas: Neural-backed generators for program synthesis. *Proceedings of the ACM on Programming Languages (OOPSLA)*.

Beaulieu-Jones, B. K., Wu, Z. S., Williams, C., Lee, R., Bhavnani, S. P., Byrd, J. B., and Greene, C. S. (2019). Privacy-preserving generative deep neural networks support clinical data sharing. *Circulation: Cardiovascular Quality and Outcomes*.

Belanger, D., Sheldon, D., and McCallum, A. (2013). Marginal inference in MRFs using Frank-Wolfe. In *NIPS Workshop on Greedy Optimization, Frank-Wolfe and Friends*.

Bertasius, G., Liu, Q., Torresani, L., and Shi, J. (2017). Local Perturb-and-MAP for structured prediction. In *International Conference on Artificial Intelligence and Statistics*.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*.

Blum, A., Ligett, K., and Roth, A. (2008). A learning theory approach to non-interactive database privacy. In *40th ACM Symposium on Theory of Computing*.

Bošnjak, M., Rocktäschel, T., Naradowsky, J., and Riedel, S. (2017). Programming with a differentiable forth interpreter. In *International Conference on Machine Learning*.

Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *Siam Review*.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599*.

Bubeck, S., Munos, R., and Stoltz, G. (2009). Pure exploration in multi-armed bandits problems. In *International conference on Algorithmic learning theory*.

Bui, T., Hernández-Lobato, D., Hernandez-Lobato, J., Li, Y., and Turner, R. (2016). Deep Gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning*.

Bunel, R. R., Desmaison, A., Mudigonda, P. K., Kohli, P., and Torr, P. (2016). Adaptive neural compilation. In *Advances in Neural Information Processing Systems*.

Burges, C. J. C. (1996). Simplified support vector decision rules. In *International Conference on Machine Learning*.

Casella, G. and Berger, R. (2002). *Statistical inference*, volume 2.

Chao, A. (1984). Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of statistics*.

Chaudhuri, K., Monteleoni, C., and Sarwate, A. D. (2011). Differentially private empirical risk minimization. *Journal of Machine Learning Research*.

Chen, X., Liu, C., and Song, D. (2019). Execution-guided neural program synthesis. In *International Conference on Learning Representations*.

Chen, Y. and Ghahramani, Z. (2016). Scalable discrete sampling as a multi-armed bandit problem. In *International Conference on Machine Learning*.

Chen, Y., Welling, M., and Smola, A. (2010). Super-samples from kernel herding. In *Uncertainty in Artificial Intelligence*.

Clymo, J., Manukian, H., Fijalkow, N., Gascón, A., and Paige, B. (2019). Data generation for neural programming by example. *arXiv:1911.02624*.

Cox, D. and Oakes, D. (1984). *Analysis of survival data*, volume 21.

Crouch, M., McGregor, A., Valiant, G., and Woodruff, D. P. (2016). Stochastic streams: Sample complexity vs. space complexity. In *European Symposium on Algorithms*.

Csiszár, I. (1995). Generalized cutoff rates and Rényi's information measures. *IEEE Transactions on information theory*.

Cui, J., Gu, M., Kwek, L. C., Santos, M. F., Fan, H., and Vedral, V. (2012). Quantum phases with differing computational power. *Nature communications*.

Cypher, A. and Halbert, D. C. (1993). *Watch what I do: programming by demonstration*.

Dankar, F. K. and El Emam, K. (2013). Practicing differential privacy in health care: A review. *Transactions on Data Privacy*.

Darbon, J. (2009). Global optimization for first order Markov random fields with submodular priors. *Discrete Applied Mathematics*.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The Helmholtz machine. *Neural computation*.

Dearden, R., Friedman, N., and Russell, S. (1998). Bayesian Q-learning. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*.

Dembczyński, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2012). On label dependence and loss minimization in multi-label classification. *Machine Learning*.

Dembczynski, K. J., Cheng, W., and Hüllermeier, E. (2010). Bayes optimal multilabel classification via probabilistic classifier chains. In *International Conference on Machine Learning*.

Dennis Jr, J. E. and Schnabel, R. B. (1996). *Numerical methods for unconstrained optimization and nonlinear equations*.

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A. R., and Kohli, P. (2017). Robustfill: Neural program learning under noisy I/O. In *International Conference on Machine Learning*.

Dwork, C. (2006). Differential privacy. In *International Conference on Automata, Languages and Programming*.

Dwork, C. and Roth, A. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*.

Dziugaite, G. K. and Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Uncertainty in Artificial Intelligence*.

Dziugaite, G. K., Roy, D. M., and Ghahramani, Z. (2015). Training generative neural networks via maximum mean discrepancy optimization. In *Uncertainty in Artificial Intelligence*.

Efron, B. and Thisted, R. (1976). Estimating the number of unseen species: How many words did Shakespeare know? *Biometrika*.

Ellis, K., Nye, M., Pu, Y., Sosa, F., Tenenbaum, J., and Solar-Lezama, A. (2019). Write, execute, assess: Program synthesis with a REPL. In *Advances in Neural Information Processing Systems*.

Ermon, S., Sabharwal, A., and Selman, B. (2013). Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *International Conference on Machine Learning*.

Feser, J. K., Chaudhuri, S., and Dillig, I. (2015). Synthesizing data structure transformations from input-output examples. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*.

Fiat, A., Karp, R. M., Luby, M., McGeoch, L. A., Sleator, D. D., and Young, N. E. (1991). Competitive paging algorithms. *Journal of Algorithms*.

Fisher, R. A. and Tippett, L. H. C. (1928). Limiting forms of the frequency distribution of the largest or smallest member of a sample. In *Mathematical Proceedings of the Cambridge Philosophical Society*.

Fredrikson, M., Jha, S., and Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM SIGSAC Conference on Computer and Communications Security*.

Friedman, W. F. (1922). *The index of coincidence and its applications in cryptology*.

Fukumizu, K., Gretton, A., Sun, X., and Schölkopf, B. (2008). Kernel measures of conditional dependence. In *Advances in Neural Information Processing Systems*.

Gaunt, A. L., Brockschmidt, M., Singh, R., Kushman, N., Kohli, P., Taylor, J., and Tarlow, D. (2016). Terpret: A probabilistic programming language for program induction. *arXiv:1608.04428*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*.

Goodman, N. D., Tenenbaum, J. B., and Gerstenberg, T. (2014). Concepts in a probabilistic language of thought. Technical report, Center for Brains, Minds and Machines (CBMM).

Goria, M. N., Leonenko, N. N., Mergel, V. V., and Novi Inverardi, P. L. (2005). A new class of random vector entropy estimators and its applications in testing statistical hypotheses. *Journal of Nonparametric Statistics*.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural Turing machines. *arXiv:1410.5401*.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*.

Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. (2015). Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*.

Gretton, A., Bousquet, O., Smola, A., and Schölkopf, B. (2005). Measuring statistical dependence with Hilbert-Schmidt norms. In *International Conference on Algorithmic Learning Theory*.

Gulwani, S. (2011). Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*.

Gulwani, S. (2016). Programming by examples: Applications, algorithms, and ambiguity resolution. In *International Joint Conference on Automated Reasoning*.

Gulwani, S., Harris, W. R., and Singh, R. (2012). Spreadsheet data manipulation using examples. *Communications of the ACM*.

Gulwani, S., Jha, S., Tiwari, A., and Venkatesan, R. (2011). Synthesis of loop-free programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*.

Gulwani, S., Polozov, O., and Singh, R. (2017). Program synthesis. *Foundations and Trends® in Programming Languages*.

Gumbel, E. J. (1935). Les valeurs extrêmes des distributions statistiques. In *Annales de l'institut Henri Poincaré*.

Gurland, J. (1956). An inequality satisfied by the Gamma function. *Scandinavian Actuarial Journal*.

Hall, R., Rinaldo, A., and Wasserman, L. (2013). Differential privacy for functions and functional data. *Journal of Machine Learning Research*.

Hao, Y. and Orlitsky, A. (2019). The broad optimality of profile maximum likelihood. *arXiv:1906.03794*.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*.

Harvey, N. J., Nelson, J., and Onak, K. (2008). Sketching and streaming entropy via approximation theory. In *49th Annual IEEE Symposium on Foundations of Computer Science*.

Hazan, T. and Jaakkola, T. (2012). On the partition function and random maximum a-posteriori perturbations. In *International Conference on Machine Learning*.

Hazan, T., Maji, S., and Jaakkola, T. (2013). On sampling from the Gibbs distribution with random maximum a-posteriori perturbations. In *Advances in Neural Information Processing Systems*.

Hazan, T., Orabona, F., Sarwate, A. D., Maji, S., and Jaakkola, T. S. (2019). High dimensional inference with random maximum a-posteriori perturbations. *IEEE Transactions on Information Theory*.

Heess, N., Tarlow, D., and Winn, J. (2013). Learning to pass expectation propagation messages. In *Advances in Neural Information Processing Systems*.

Hennessy, J. L. and Patterson, D. A. (2019). A new golden age for computer architecture. *ACM*.

Hero, A. O., Bing Ma, Michel, O. J. J., and Gorman, J. (2002). Applications of entropic spanning graphs. *IEEE Signal Processing Magazine*.

Hero, A. O. and Michel, O. J. (1999). Asymptotic theory of greedy approximations to minimal k-point random graphs. *IEEE Transactions on Information Theory*.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A. R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., et al. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*.

Indyk, P. and Woodruff, D. (2005). Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th annual ACM symposium on Theory of computing*.

Jampani, V., Nowozin, S., Loper, M., and Gehler, P. V. (2015). The informed sampler: A discriminative approach to Bayesian inference in generative computer vision models. *Computer Vision and Image Understanding*.

Jenssen, R., Hild, K., Erdogmus, D., Principe, J. C., and Eltoft, T. (2003). Clustering using Rényi's entropy. In *International Joint Conference on Neural Networks*.

Ji, Z. and Elkan, C. (2013). Differential privacy based on importance weighting. *Machine learning*.

Jordan, M. I. (2018). On gradient-based optimization: Accelerated, stochastic and nonconvex. Microsoft Research.

Jordan, M. I. (2019). Artificial intelligence—the revolution hasn't happened yet. *Harvard Data Science Review*.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*.

Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*.

Kaiser, Ł. and Sutskever, I. (2016). Neural GPUs learn algorithms. In *International Conference on Learning Representations*.

Kanagawa, M., Hennig, P., Sejdinovic, D., and Sriperumbudur, B. (2018). Gaussian processes and kernel methods: A review on connections and equivalences. *arXiv:1807.02582*.

Kar, P. and Karnick, H. (2012). Random feature maps for dot product kernels. In *International Conference on Artificial Intelligence and Statistics*.

Kelley, H. J. (1960). Gradient theory of optimal flight paths. *Ars Journal*.

Kenthapadi, K., Korolova, A., Mironov, I., and Mishra, N. (2012). Privacy via the Johnson-Lindenstrauss transform. *arXiv:1204.2606*.

Kim, C., Sabharwal, A., and Ermon, S. (2016). Exact sampling with integer linear programs and random perturbations. In *AAAI Conference on Artificial Intelligence*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2014). Stochastic gradient VB and the variational auto-encoder. In *International Conference on Learning Representations*.

Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *IEEE transactions on pattern analysis and machine intelligence*.

Kowalczyk, L., Malkin, T., Ullman, J., and Wichs, D. (2018). Hardness of non-interactive differential privacy from one-way functions. In *Annual International Cryptology Conference*. Springer.

Krishnan, R. G., Lacoste-Julien, S., and Sontag, D. (2015). Barrier Frank-Wolfe for marginal inference. In *Advances in Neural Information Processing Systems*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Kurach, K., Andrychowicz, M., and Sutskever, I. (2016). Neural random-access machines. In *International Conference on Learning Representations*.

Lagrange, J.-L. (1788). *Mécanique Analytique*.

Laich, L., Bielik, P., and Vechev, M. (2020). Guiding program synthesis by learning to generate examples. In *International Conference on Learning Representations*.

Lample, G. and Charton, F. (2020). Deep learning for symbolic mathematics. In *International Conference on Learning Representations*.

Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted Boltzmann machines. In *International Conference on Machine Learning*.

Lauritzen, S. (1996). *Graphical models*. Oxford.

Le, T. A., Baydin, A. G., Zinkov, R., and Wood, F. (2017). Using synthetic data to train neural networks is model-based reasoning. In *International Joint Conference on Neural Networks*.

Learned-Miller, E. G. and John III, W. F. (2003). ICA using spacings estimates of entropy. *Journal of Machine Learning Research*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K. R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*.

Lenz, W. (1920). Beitrag zum verständnis der magnetischen erscheinungen in festen körpern. *Physikalische Zeitschrift*.

Li, C.-L., Chang, W.-C., Cheng, Y., Yang, Y., and Póczos, B. (2017). MMD GAN: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*.

Li, Y. and Gal, Y. (2017). Dropout inference in Bayesian neural networks with alpha-divergences. In *International Conference on Machine Learning*.

Li, Y., Swersky, K., and Zemel, R. (2015). Generative moment matching networks. In *International Conference on Machine Learning*.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. (2016). Gated graph sequence neural networks. In *International Conference on Learning Representations*.

Li, Y. and Turner, R. E. (2016). Rényi divergence variational inference. In *Advances in Neural Information Processing Systems*.

Liang, P. (2016). Learning executable semantic parsers for natural language understanding. *Communications of the ACM*.

Lin, X. V., Wang, C., Pang, D., Vu, K., and Ernst, M. D. (2017). Program synthesis from natural language using recurrent neural networks. *University of Washington Department of Computer Science and Engineering*.

Ling, W., Grefenstette, E., Hermann, K. M., Kočiský, T., Senior, A., Wang, F., and Blunsom, P. (2016). Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Little, W. A. (1974). The existence of persistent states in the brain. In *From High-Temperature Superconductivity to Microminiature Refrigeration*.

Loos, S., Irving, G., Szegedy, C., and Kaliszyk, C. (2017). Deep network guided proof search. *arXiv:1701.06972*.

Lopez-Paz, D., Muandet, K., Schölkopf, B., and Tolstikhin, I. (2015). Towards a learning theory of cause-effect inference. In *International Conference on Machine Learning*.

Ma, B., Hero, A., Gorman, J., and Michel, O. (2000). Image registration with minimum spanning tree algorithm. In *International Conference on Image Processing*.

Machanavajjhala, A., Kifer, D., Abowd, J., Gehrke, J., and Vilhuber, L. (2008). Privacy: Theory meets practice on the map. In *IEEE 24th International Conference on Data Engineering*.

Maddison, C. (2016). A Poisson process model for Monte Carlo. In Hazan, T., Papandreou, G., and Tarlow, D., editors, *Perturbation, Optimization, and Statistics*.

Maddison, C., Tarlow, D., and Minka, T. (2014). A* sampling. In *Advances in Neural Information Processing Systems*.

Maji, S., Hazan, T., and Jaakkola, T. (2014). Active boundary annotation using random MAP perturbations. In *International Conference on Artificial Intelligence and Statistics*.

McSherry, F. and Talwar, K. (2007). Mechanism design via differential privacy. In *IEEE Symposium on Foundations of Computer Science*.

Menon, A. K., Tamuz, O., Gulwani, S., Lampson, B. W., and Kalai, A. (2013). A machine learning framework for programming by example. In *International Conference on Machine Learning*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*.

Minka, T. (2005). Divergence measures and message passing. Technical report.

Minka, T. P. (2001). Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*.

Mooij, J. (2010). libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*.

Muandet, K., Sriperumbudur, B., Fukumizu, K., Gretton, A., and Schölkopf, B. (2016). Kernel mean shrinkage estimators. *Journal of Machine Learning Research*.

Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in Artificial Intelligence*.

Neelakantan, A., Le, Q. V., and Sutskever, I. (2016). Neural programmer: Inducing latent programs with gradient descent. In *International Conference on Learning Representations*.

Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate o (1/k^2). In *Dokl. Akad. Nauk SSSR*.

Nisan, N. (1996). Extracting randomness: how and why. a survey. In *Proceedings of Computational Complexity*.

Nye, M., Hewitt, L., Tenenbaum, J., and Solar-Lezama, A. (2019). Learning to infer program sketches. In *International Conference on Machine Learning*.

Obremski, M. and Skorski, M. (2017). Rényi entropy estimation revisited. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*.

Orabona, F., Hazan, T., Sarwate, A., and Jaakkola, T. (2014). On measure concentration of random maximum a-posteriori perturbations. In *International Conference on Machine Learning*.

Orlitsky, A., Santhanam, N. P., Viswanathan, K., and Zhang, J. (2004). On modeling profiles instead of values. In *Uncertainty in Artificial Intelligence*.

Orlitsky, A., Suresh, A. T., and Wu, Y. (2016). Optimal prediction of the number of unseen species. *Proceedings of the National Academy of Sciences*.

Pál, D., Póczos, B., and Szepesvári, C. (2010). Estimation of Rényi entropy and mutual information based on generalized nearest-neighbor graphs. In *Advances in Neural Information Processing Systems*.

Paninski, L. (2003). Estimation of entropy and mutual information. *Neural computation*.

Papandreou, G. and Yuille, A. (2010). Gaussian sampling by local perturbations. In *Advances in Neural Information Processing Systems*.

Papandreou, G. and Yuille, A. (2011). Perturb-and-MAP random fields: Using discrete optimization to learn and sample from energy models. In *International Conference on Computer Vision*.

Parisi, G. (1988). *Statistical Field Theory*. Frontiers in Physics.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*.

Pham, D.-T. (2004). Fast algorithms for mutual information based independent component analysis. *IEEE Transactions on Signal Processing*.

Piech, C., Huang, J., Nguyen, A., Phulsuksombati, M., Sahami, M., and Guibas, L. J. (2015). Learning program embeddings to propagate feedback on student code. In *International Conference on Machine Learning*.

Polosukhin, I. and Skidanov, A. (2018). Neural program search: Solving data processing tasks from description and examples. *ICLR Workshop Track*.

Polozov, O. and Gulwani, S. (2015). FlashMeta: a framework for inductive program synthesis. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

Porta, A., Guzzetti, S., Montano, N., Furlan, R., Pagani, M., Malliani, A., and Cerutti, S. (2001). Entropy, entropy rate, and pattern classification as tools to typify complexity in short heart period variability series. *IEEE Transactions on Biomedical Engineering*.

Potts, R. B. (1952). Some generalized order-disorder transformations. *Mathematical Proceedings of the Cambridge Philosophical Society*.

Principe, J. C. and Xu, D. (1999). Information-theoretic learning using Rényi's quadratic entropy. In *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation*.

Rahimi, A. and Recht, B. (2007). Random features for large scale kernel machines. In *Advances in Neural Information Processing Systems*.

Raj, A., Law, H. C. L., Sejdinovic, D., and Park, M. (2019). A differentially private kernel two-sample test. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning*.

Reed, S. and De Freitas, N. (2016). Neural programmer-interpreters. In *International Conference on Learning Representations*.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*.

Rubinstein, B. I. P., Bartlett, P. L., Huang, L., and Taft, N. (2012). Learning in a large function space: Privacy-preserving mechanisms for SVM learning. *The Journal of Privacy and Confidentiality*.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*.

Rényi, A. (1960). On measures of information and entropy. In *Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability*.

Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *Artificial intelligence and statistics*.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *International Conference on Machine Learning*.

Santha, M. and Vazirani, U. V. (1986). Generating quasi-random sequences from semi-random sources. *Journal of computer and system sciences*.

Schkufza, E., Sharma, R., and Aiken, A. (2016). Stochastic program optimization. *Commununications of the ACM*.

Schölkopf, B., Muandet, K., Fukumizu, K., Harmeling, S., and Peters, J. (2015). Computing functions of random variables via Reproducing Kernel Hilbert Space representations. *Statistics and Computing*.

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. (2015). Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*.

Shin, E. C., Allamanis, M., Brockschmidt, M., and Polozov, A. (2019a). Program synthesis and semantic parsing with learned code idioms. In *Advances in Neural Information Processing Systems*.

Shin, R., Kant, N., Gupta, K., Bender, C., Trabucco, B., Singh, R., and Song, D. (2019b). Synthetic datasets for neural program synthesis. In *International Conference on Learning Representations*.

Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., and Moore, R. (2013). Real-time human pose recognition in parts from single depth images. *Communications of the ACM*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*.

Simon-Gabriel, C.-J., Ścibior, A., Tolstikhin, I., and Schölkopf, B. (2016). Consistent kernel mean estimation for functions of random variables. In *Advances in Neural Information Processing Systems*.

Singh, R. and Gulwani, S. (2015). Predicting a correct program in programming by example. In *Proceedings of the 27th Conference on Computer Aided Verification*.

Smola, A., Gretton, A., Song, L., and Schölkopf, B. (2007). A Hilbert space embedding for distributions. In *International Conference on Algorithmic Learning Theory*.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science.

Solar-Lezama, A. (2008). *Program Synthesis By Sketching*. PhD thesis, EECS Dept., UC Berkeley.

Sontag, D., Meltzer, T., Globerson, A., Jaakkola, T. S., and Weiss, Y. (2008). Tightening LP relaxations for MAP using message passing. In *Uncertainty in Artificial Intelligence*.

Sriperumbudur, B. and Szabo, Z. (2015). Optimal rates for random Fourier features. In *Advances in Neural Information Processing Systems*.

Sriperumbudur, B. K., Fukumizu, K., and Lanckriet, G. R. G. (2011). Universality, characteristic kernels and RKHS embedding of measures. *Journal of Machine Learning Research*.

Stéphan, J.-M., Misguich, G., and Pasquier, V. (2010). Rényi entropy of a line in two-dimensional Ising models. *Physical Review B*.

Stuhlmüller, A., Taylor, J., and Goodman, N. D. (2013). Learning stochastic inverses. In *Advances in Neural Information Processing Systems*.

Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. (2015). End-to-end memory networks. In *Advances in Neural Information Processing Systems*.

Sutskever, I. and Hinton, G. (2007). Learning multilevel distributed representations for high-dimensional sequences. In *Artificial intelligence and statistics*.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*.

Tarlow, D., Adams, R., and Zemel, R. (2012). Randomized optimum models for structured prediction. In *International Conference on Artificial Intelligence and Statistics*.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*.

Tolstikhin, I., Sriperumbudur, B. K., and Muandet, K. (2017). Minimax estimation of kernel mean embeddings. *Journal of Machine Learning Research*.

Valiant, G. and Valiant, P. (2011). Estimating the unseen: an n/log (n)-sample estimator for entropy and support size, shown optimal via new clts. In *ACM symposium on Theory of computing*.

Valiant, L. (1979). The complexity of computing the permanent. *Theoretical Computer Science*.

Vasicek, O. (1976). A test for normality based on sample entropy. *Journal of the Royal Statistical Society*.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*.

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*.

Wasserman, L. and Zhou, S. (2010). A statistical framework for differential privacy. *Journal of the American Statistical Association*.

Weller, A. and Domke, J. (2016). Clamping improves TRW and mean field approximations. In *International Conference on Artificial Intelligence and Statistics*.

Weller, A. and Jebara, T. (2014a). Approximating the Bethe partition function. In *Uncertainty in Artificial Intelligence*.

Weller, A. and Jebara, T. (2014b). Clamping variables and approximate inference. In *Advances in Neural Information Processing Systems*.

Weston, J., Chopra, S., and Bordes, A. (2015). Memory networks. In *International Conference on Learning Representations*.

Wright, S. and Nocedal, J. (1999). Numerical optimization. *Springer Science*.

Wu, Y. and Yang, P. (2016). Minimax rates of entropy estimation on large alphabets via best polynomial approximation. *IEEE Transactions on Information Theory*.

Zaremba, W., Mikolov, T., Joulin, A., and Fergus, R. (2016). Learning simple algorithms from examples. In *International Conference on Machine Learning*.

Zhang, C., Butepage, J., Kjellstrom, H., and Mandt, S. (2018). Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*.

Zhao, J., Djolonga, J., Tschiatschek, S., and Krause, A. (2016). Variable clamping for optimization-based inference. In *NIPS Workshop on Advances in Approximate Bayesian Inference*.

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv:1709.00103*.

Zhou, S., Ligett, K., and Wasserman, L. (2009). Differential privacy with compression. In *IEEE International Conference on Symposium on Information Theory*.

Zohar, A. and Wolf, L. (2018). Automatic program synthesis of long programs with a learned garbage collector. In *Advances in Neural Information Processing Systems*.

Appendix A

Partition Function Estimation

Here we provide proofs for the results stated in Chapter 2, together with additional supporting lemmas required for these proofs.

A.1 Comparison of Gumbel and Exponential Tricks

In Section 2.2.3 we analyzed the asymptotic efficiency of different estimators of Z by measuring their asymptotic variance. (As all our estimators in the full-rank perturbation setting are consistent, their bias is 0 in the limit of infinite data, and so this asymptotic variance equals the asymptotic MSE.) In the non-asymptotic regime, where an estimator $\hat{Z}$ is constructed from a finite set of M samples, we can analyze both the variance $\mathrm{var}(\hat{Z})$ and the bias $(\mathbb{E}[\hat{Z}] - Z)$ of the estimator. While in most cases these cannot be obtained analytically and there we can resort to an empirical evaluation, for the estimators stemming from the Gumbel and Exponential tricks analytical treatment turns out to be possible using standard methods.

A.1.1 Estimating Z

Gumbel trick The Gumbel trick yields an unbiased estimator for $\ln Z$, and we can turn it into a consistent estimator of Z by exponentiating it:

$$\hat{Z} := \exp\left(\frac{1}{M}\sum_{m=1}^{M} X_m\right) \qquad \text{where} \qquad X_1, \ldots, X_M \overset{\text{iid}}{\sim} \text{Gumbel}(-c + \ln Z).$$

Recalling that the moment generating function of a Gumbel(μ) distribution is $G(t) = \Gamma(1-t)e^{\mu t}$, we can obtain by using independence of the samples:

$$\mathbb{E}[\hat{Z}] = \prod_{m=1}^{M} \mathbb{E}[e^{X_m/M}] = \left(\Gamma(1-1/M)e^{(\ln Z - c)/M}\right)^M = \Gamma(1-1/M)^M e^{-c} Z,$$
$$\mathbb{E}[\hat{Z}^2] = \prod_{m=1}^{M} \mathbb{E}[e^{2X_m/M}] = \left(\Gamma(1-2/M)e^{2(\ln Z - c)/M}\right)^M = \Gamma(1-2/M)^M e^{-2c} Z^2.$$

Therefore the squared bias, variance and MSE of the estimator $\hat{Z}$ are, respectively:

$$\begin{aligned}
\text{bias}(\hat{Z})^2 &= (\mathbb{E}[\hat{Z}] - Z)^2 = Z^2\left(\Gamma(1-1/M)^M e^{-c} - 1\right), \\
\text{var}(\hat{Z}) &= \mathbb{E}[\hat{Z}^2] - \mathbb{E}[\hat{Z}]^2 = Z^2\left(\Gamma(1-2/M)^M e^{-2c} - \Gamma(1-1/M)^{2M} e^{-2c}\right), \\
\text{MSE}(\hat{Z}) &= \text{bias}(\hat{Z})^2 + \text{var}(\hat{Z}) = Z^2\left(\Gamma(1-2/M)^M e^{-2c} - 2\Gamma(1-1/M)^M e^{-c} + 1\right).
\end{aligned}$$

These formulas hold for $M > 2$ where the moment generating functions are defined. For $M = 1$ the estimator has infinite bias (and infinite variance), and for $M = 2$ it has infinite variance. Figure 2.1 (left) shows the functional dependence of MSE($\hat{Z}$) on the number of samples $M \geq 3$, in units of Z^2.

Exponential trick The Exponential trick yields an unbiased estimator of $1/Z$, and we can turn it into a consistent estimator of Z by inverting it:

$$\hat{Z} := \left(\frac{1}{M}\sum_{m=1}^{M} X_m\right)^{-1} \qquad \text{where} \qquad X_1, \ldots, X_M \overset{\text{iid}}{\sim} \text{Exp}(Z).$$

As $X_1, \ldots, X_M$ are independent and exponentially distributed with identical rates Z, their sum follows the Gamma distribution with shape M and rate Z. Therefore the estimator $\hat{Z}$ can be written as $\hat{Z} = MY$, where $Y \sim \text{InvGamma}(M, Z)$. Recalling the mean and variance of the Inverse-Gamma distribution, we obtain:

$$\begin{aligned}
\text{bias}(\hat{Z})^2 &= (\mathbb{E}[\hat{Z}] - Z)^2 = Z^2\left(\frac{M}{M-1} - 1\right) = Z^2\frac{1}{M-1}, \\
\text{var}(\hat{Z}) &= Z^2 M^2 \frac{1}{(M-1)^2(M-2)}, \\
\text{MSE}(\hat{Z}) &= \text{bias}(\hat{Z})^2 + \text{var}(\hat{Z}) = Z^2\frac{M-2+M^2}{(M-1)^2(M-2)} = Z^2\frac{M+2}{(M-1)(M-2)}.
\end{aligned}$$

Again these formulas hold for $M > 2$ where the relevant expectations are defined: for $M = 1$ the estimator has infinite bias, and for $M \in \{1, 2\}$ it has infinite variance. Figure 2.1 (left) shows the functional dependence of $\mathrm{MSE}(\hat{Z})$ on the number of samples $M \geq 3$, in units of Z^2. By inspecting the curves we observe that the Gumbel trick estimator requires roughly 45% more samples to yield the same MSE as the Exponential trick estimator.

A.1.2 Estimating $\ln Z$

A similar analysis can be performed for estimating $\ln Z$ rather than Z. In that case the Gumbel trick estimator of $\ln Z$ is unbiased and has variance (and thus MSE) equal to $\frac{1}{M}\frac{\pi^2}{6}$. On the other hand, the Exponential trick estimator is

$$\widehat{\ln Z} = -\ln\left(\frac{1}{M}\sum_{m=1}^{M} X_m\right) \qquad \text{where} \qquad X_1, \ldots, X_M \overset{\text{iid}}{\sim} \mathrm{Exp}(Z).$$

Again $\sum_{m=1}^{M} X_m \sim \mathrm{Gamma}(M, Z)$ and by reference to properties of the Gamma distribution,

$$\begin{aligned}
\mathrm{bias}(\widehat{\ln Z})^2 &= (\mathbb{E}[\hat{Z}] - Z)^2 = (\ln M - (\psi(M) - \ln Z) - \ln Z)^2 = (\ln M - \psi(M))^2, \\
\mathrm{var}(\widehat{\ln Z}) &= \psi_1(M), \\
\mathrm{MSE}(\widehat{\ln Z}) &= \mathrm{bias}(\widehat{\ln Z})^2 + \mathrm{var}(\widehat{\ln Z}) = (\ln M - \psi(M))^2 + \psi_1(M),
\end{aligned}$$

where $\psi(\cdot)$ is the digamma function and $\psi_1(\cdot)$ is the trigamma function. Note that the estimator can be debiased by subtracting its bias $(\ln M - \psi(M))$. Figure 2.1 (right) compares the MSE of the Gumbel and Exponential trick estimators of $\ln Z$. We observe that the Gumbel trick estimator performs better only for $M = 1$, and even in that case the Exponential trick estimator is better when debiased.

A.2 Sum-unary Perturbations

Recall that *sum-unary perturbations* refer to the setting where each variable's unary potentials are perturbed with Gumbel noise, and the perturbed potential of a configuration sums the perturbations from all variables (see Definition 1 in the main text). Using sum-unary perturbations we can derive a family $\mathcal{U}(\alpha)$ of upper bounds on the log partition function (Proposition 2) and construct sequential samplers for the Gibbs

distribution (Algorithm 1). Here we provide proofs for the related results stated in Sections 2.3.1 and 2.3.2.

Notation We will write $\operatorname{pow}_\beta x$ for x^β, where $x, \beta \in \mathbb{R}$, when we find this increases clarity of our exposition.

Lemma 3 (Weibull and Fréchet tricks)**.** *For any finite set $\mathcal{Y}$ and any function h, we have*

$$\operatorname*{pow}_{-\alpha} \sum_{y\in\mathcal{Y}} \operatorname*{pow}_{-1/\alpha} h(y) = \mathbb{E}_W \left[\min_y \left\{ h(y) \frac{W(y)}{\Gamma(1+\alpha)} \right\} \right]$$

where $\{W(y)\}_{y\in\mathcal{Y}} \overset{iid}{\sim} \text{Weibull}(1, \alpha^{-1})$ for $\alpha \in (0, \infty)$, and

$$\operatorname*{pow}_{-\alpha} \sum_{y\in\mathcal{Y}} \operatorname*{pow}_{-1/\alpha} h(y) = \mathbb{E}_F \left[\max_y \left\{ h(y) \frac{F(y)}{\Gamma(1+\alpha)} \right\} \right]$$

where $\{F(y)\}_{y\in\mathcal{Y}} \overset{i.i.d.}{\sim} \text{Fréchet}(1, -\alpha^{-1})$ for $\alpha \in (-1, 0)$.

Proof. This follows from setting up competing exponential clocks with rates $\lambda_y = h(y)^{-1/\alpha}$ and then applying the function $g(x) = x^\alpha$ as in Example 6 for the case of the Weibull trick. The case of the Fréchet trick is similar, except that g is strictly decreasing for $\alpha \in (-1, 0)$, hence the maximization in place of the minimization. □

A.2.1 Upper Bounds on the Partition Function

Proposition 2. For any $\alpha \in (-1, 0) \cup (0, \infty)$, the upper bound $\ln Z \leq \mathcal{U}(\alpha)$ holds with

$$\mathcal{U}(\alpha) := n \frac{\ln \Gamma(1+\alpha)}{\alpha} + nc - \frac{1}{\alpha} \ln \mathbb{E}_\gamma \left[e^{-\alpha U} \right].$$

Proof. We show the result for $\alpha \in (0, \infty)$ using the Weibull trick; the case of $\alpha \in (-1, 0)$ can be proved similarly using the Fréchet trick. The idea is to prove by induction on n that $Z^{-\alpha} \geq e^{-\alpha \mathcal{U}(\alpha)}$, so that the claimed result follows by applying the monotonically decreasing function $x \mapsto -\ln(x)/\alpha$.

The base case $n = 1$ is the Clamping Lemma 1 below with $j = n = 1$. Now assume the claim for $n - 1 \geq 1$ and for $x_n \in \mathcal{X}_n$ define $\mathcal{U}_{n-1}(\alpha, x_1) :=$

$$(n-1) \frac{\ln \Gamma(1+\alpha)}{\alpha} + (n-1)c - \frac{1}{\alpha} \ln \mathbb{E}_\gamma \left[\exp \left(-\alpha \max_{x_2,\ldots,x_n} \left\{ \phi(x) + \sum_{i=2}^{n} \gamma_i(x_i) \right\} \right) \right].$$

With this definition, the Clamping Lemma with $j = 1$ states that

$$\sum_{x_1} \underset{-1/\alpha}{\operatorname{pow}} e^{-\alpha \mathcal{U}_{n-1}(\alpha, x_1)} \leq \underset{-1/\alpha}{\operatorname{pow}} e^{-\alpha \mathcal{U}(\alpha)},$$

so

$$\begin{aligned} Z^{-\alpha} &\geq \underset{-\alpha}{\operatorname{pow}} \sum_{x_1 \in \mathcal{X}_1} \underset{-1/\alpha}{\operatorname{pow}} e^{-\alpha \mathcal{U}_{n-1}(\alpha, x_1)} && \text{[inductive hypothesis]} \\ &\geq \underset{-\alpha}{\operatorname{pow}} \underset{-1/\alpha}{\operatorname{pow}} e^{-\alpha \mathcal{U}(\alpha)} && \text{[Clamping Lemma]} \\ &= e^{-\alpha \mathcal{U}(\alpha)}, \end{aligned}$$

as required to complete the inductive step. □

Proposition 3. The limit of $\mathcal{U}(\alpha)$ as $\alpha \to 0$ exists and equals $\mathcal{U}(0) := \mathbb{E}[U]$, i.e. the Gumbel trick upper bound.

Proof. Recall that $\mathcal{U}(\alpha) = n\frac{\ln \Gamma(1+\alpha)}{\alpha} + nc - \frac{1}{\alpha} \ln \mathbb{E}\left[e^{-\alpha U}\right]$. The first term tends to $n\psi(1) = -cn$ as $\alpha \to 0$ by L'Hôpital's rule, where ψ is the digamma function. The second term is constant in α. In the last term, $\mathbb{E}\left[e^{-\alpha U}\right]$ is the moment generating function of U evaluated at $-\alpha$, and as such its derivative at $\alpha = 0$ exists and equals the negative of the mean of U. Hence by L'Hôpital's rule,

$$-\lim_{\alpha \to 0} \frac{1}{\alpha} \ln \mathbb{E}\left[e^{-\alpha U}\right] = -\lim_{\alpha \to 0} \frac{-\mathbb{E}[U]}{\mathbb{E}\left[e^{-\alpha U}\right]} = \mathbb{E}[U] = \mathcal{U}(0).$$

The claimed result then follows by the Algebra of Limits, as the contributions of the first two terms cancel. □

Proposition 4. The function $\mathcal{U}(\alpha)$ is differentiable at $\alpha = 0$ and the derivative equals

$$\frac{\mathrm{d}}{\mathrm{d}\alpha} \mathcal{U}(\alpha)\bigg|_{\alpha=0} = n\frac{\pi^2}{12} - \frac{\operatorname{var}(U)}{2}.$$

Proof. First we show that $\mathcal{U}(\alpha)$ is differentiable on $(-1, 0) \cup (0, \infty)$, and that the limit of the derivative as $\alpha \to 0$ exists and equals $n\pi^2/12 - \operatorname{var}(U)/2$.

The first term of $\mathcal{U}(\alpha)$ is $n\frac{\ln \Gamma(1+\alpha)}{\alpha}$, which is differentiable for $\alpha \in (-1, 0) \cup (0, \infty)$ by the Quotient Rule, and its derivative equals

$$\frac{\mathrm{d}}{\mathrm{d}\alpha} n\frac{\ln \Gamma(1+\alpha)}{\alpha} = n\frac{\psi(1+\alpha)\alpha - \ln \Gamma(1+\alpha)}{\alpha^2},$$

where ψ is the digamma function (logarithmic derivative of the gamma function). Applying L'Hôpital's rule we note that

$$\begin{aligned}\lim_{\alpha\to 0}\frac{\mathrm{d}}{\mathrm{d}\alpha}n\frac{\ln\Gamma(1+\alpha)}{\alpha} &= n\lim_{\alpha\to 0}\frac{\psi(1+\alpha)+\alpha\psi^{(1)}(1+\alpha)-\psi(1+\alpha)}{2\alpha}\\ &= n\frac{\psi^{(1)}(1)}{2} = n\frac{\zeta(2)}{2} = n\frac{\pi^2}{12},\end{aligned}$$

where $\psi^{(1)}$ is the trigamma function (derivative of the digamma function), whose value at 1 is known to be $\zeta(2) = \pi^2/6$, the Riemann zeta function evaluated at 2.

The second term of $\mathcal{U}(\alpha)$ is constant in α. The last term can be written as $K(-\alpha)/(-\alpha)$, where K is the cumulant generating function (logarithm of the moment generating function) of the random variable U. The cumulant generating function is differentiable, and by the Quotient rule

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\frac{K(-\alpha)}{-\alpha} = -\frac{\alpha K'(-\alpha)-K(-\alpha)}{\alpha^2}.$$

Applying L'Hôpital's rule we note that

$$\lim_{\alpha\to 0}\frac{\mathrm{d}}{\mathrm{d}\alpha}\frac{K(-\alpha)}{-\alpha} = \lim_{\alpha\to 0}\frac{K'(-\alpha)+\alpha K''(-\alpha)-K'(-\alpha)}{2\alpha} = \frac{K''(0)}{2} = \frac{\mathrm{var}(U)}{2},$$

where we have used that the second derivative of the cumulant generating function is the variance.

As $\mathcal{U}(\alpha)$ is continuous at 0 by construction, the above implies that it has left and right derivatives at 0. As the values of these derivatives coincide, the function is differentiable at 0 and the derivative has the stated value. □

Recall that for a variable index $j \in \{1,\dots,n\}$ we also defined *partial sum-unary perturbations*

$$U_j(x_1,\dots,x_{j-1}) := \max_{x_j,\dots,x_n}\left\{\phi(\mathbf{x}) + \sum_{i=j}^{n}\gamma_i(x_i)\right\},$$

which fix the variables $x_1,\dots,x_{j-1}$ and perturb the remaining ones.

Lemma 1 (Clamping Lemma)**.** For any $j \in \{1,\dots,n\}$ and any fixed partial variable assignment $(x_1,\dots,x_{j-1}) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_{j-1}$, the following inequality holds with any

trick parameter $\alpha \in (-1, 0) \cup (0, \infty)$:

$$\sum_{x_j \in \mathcal{X}_j} \mathbb{E}_\gamma \left[e^{-(n-j)\ln\Gamma(1+\alpha) - \alpha(n-j)c} e^{-\alpha U_{j+1}(x_1,\ldots,x_j)} \right]^{-1/\alpha}$$
$$\leq \mathbb{E}_\gamma \left[e^{-(n-(j-1))\ln\Gamma(1+\alpha) - \alpha(n-(j-1))c} e^{-\alpha U_j(x_1,\ldots,x_{j-1})} \right]^{-1/\alpha}.$$

Proof. For $\alpha > 0$, from the Weibull trick (Lemma 3), using independence of the perturbations and Jensen's inequality,

$$\begin{aligned}
&\operatorname*{pow}_{-\alpha} \sum_{x_j \in \mathcal{X}_j} \operatorname*{pow}_{-1/\alpha} \mathbb{E}_W \left[\min_{x_{j+1},\ldots,x_n} \tilde{p}(\mathbf{x})^{-\alpha} \prod_{i=j+1}^{n} \frac{W(x_i)}{\Gamma(1+\alpha)} \right] \\
&= \mathbb{E}_W \left[\min_{x_j \in \mathcal{X}_j} \left\{ \mathbb{E}_W \left[\min_{x_{j+1},\ldots,x_n} \tilde{p}(\mathbf{x})^{-\alpha} \prod_{i=j+1}^{n} \frac{W(x_i)}{\Gamma(1+\alpha)} \right] \frac{W(x_j)}{\Gamma(1+\alpha)} \right\} \right] \\
&\leq \mathbb{E}_W \left[\min_{x_j,\ldots,x_n} \tilde{p}(\mathbf{x})^{-\alpha} \prod_{i=j}^{n} \frac{W(x_i)}{\Gamma(1+\alpha)} \right].
\end{aligned}$$

Representing the Weibull random variables in terms of Gumbel random variables using the transformation $W = e^{-(\gamma+c)\alpha}$, where $\gamma \sim \text{Gumbel}(-c)$, and manipulating the obtained expressions yields the claimed result. □

A.2.2 Sequential Samplers for the Gibbs Distribution

The family of sequential samplers for the Gibbs distribution presented in the main text as Algorithm 1 has the same overall structure as the sequential sampler derived by Hazan et al. (2013) from the Gumbel trick upper bound $\mathcal{U}(0)$, and hence correctness can be argued similarly. Conditioned on accepting the sample, the probability that $\mathbf{x} = (x_1, \ldots, x_n)$ is returned is

$$\begin{aligned}
\prod_{i=1}^{n} p_i(x_i | x_1, \ldots, x_{i-1}) &= \prod_{i=1}^{n} \frac{e^{-c}}{\Gamma(1+\alpha)^{1/\alpha}} \frac{\mathbb{E}_\gamma \left[e^{-\alpha U_{i+1}(x_1,\ldots,x_i)} \right]^{-1/\alpha}}{\mathbb{E}_\gamma \left[e^{-\alpha U_i(x_1,\ldots,x_{i-1})} \right]^{-1/\alpha}} \\
&= \frac{e^{-nc}}{\Gamma(1+\alpha)^{n/\alpha}} \frac{\left(e^{-\alpha\phi(x_1,\ldots,x_n)} \right)^{-1/\alpha}}{\mathbb{E}[e^{-\alpha U}]^{-1/\alpha}} \propto p(x),
\end{aligned}$$

as required to show that the produced samples follow the Gibbs distribution p. Note, however, that in practice one introduces an approximation by replacing expectations with sample averages.

A.2.3 Relationship Between Errors of Sum-unary Gumbel Perturbations

We write $\mathbf{x}^*$ for the (random) MAP configuration after sum-unary perturbation of the potential function, i.e.,

$$\mathbf{x}^* := \operatorname*{argmax}_{\mathbf{x}\in\mathcal{X}} \left\{ \phi(\mathbf{x}) + \sum_{i=1}^{n} \gamma_i(x_i) \right\}.$$

Let $q_{\text{sum}}(\mathbf{x}) := \mathbb{P}[\mathbf{x} = \mathbf{x}^*]$ be the probability mass function of $\mathbf{x}^*$.

The following results links together the errors acquired when using summed unary perturbations to upper bound the log partition function $\ln Z \leq \mathcal{U}(0)$ using the Gumbel trick upper bound by Hazan and Jaakkola (2012), to approximately sample from the Gibbs distribution by using q_{sum} instead, and to upper bound the entropy of the approximate distribution q_{sum} using the bound due to Maji et al. (2014).

Proposition 6. Writing p for the Gibbs distribution, we have

$$\underbrace{(\mathcal{U}(0) - \ln Z)}_{\text{error in } \ln Z \text{ bound}} + \underbrace{\mathrm{KL}(q_{\text{sum}} \parallel p)}_{\text{sampling error}} = \underbrace{\mathbb{E}_{\gamma_i}\left[\gamma_i(\mathbf{x}_i^*)\right] - H(q_{\text{sum}})}_{\text{error in entropy estimation}}.$$

Proof. By conditioning on the maximizing configuration $\mathbf{x}^*$, we can rewrite the Gumbel trick upper bound $\mathcal{U}(0)$ as follows:

$$\begin{aligned}
\mathcal{U}(0) &= \mathbb{E}_{\gamma}\left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \theta(\mathbf{x}) + \sum_{i=1}^{n} \gamma_i(x_i) \right\}\right] \\
&= \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{sum}}(\mathbf{x}) \left(\theta(\mathbf{x}) + \mathbb{E}_{\gamma}\left[\sum_{i=1}^{n} \gamma_i(x_i) \mid \mathbf{x} = \mathbf{x}^* \right] \right) \\
&= \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{sum}}(\mathbf{x})\theta(\mathbf{x}) + \sum_{i=1}^{n} \mathbb{E}_{\gamma_i}\left[\gamma_i(x_i^*)\right].
\end{aligned}$$

At the same time, the KL divergence between q_{sum} and the Gibbs distribution p generally expands as

$$\begin{aligned}
\mathrm{KL}(q_{\text{sum}} \parallel p) &= -H(q_{\text{sum}}) - \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{sum}}(\mathbf{x}) \ln \frac{\exp\left(\theta(\mathbf{x})\right)}{\sum_{\tilde{\mathbf{x}}\in\mathcal{X}} \exp\left(\theta(\tilde{\mathbf{x}})\right)} \\
&= -H(q_{\text{sum}}) - \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{sum}}(\mathbf{x})\theta(\mathbf{x}) + \ln Z.
\end{aligned}$$

Adding the two equations together and rearranging yields the claimed result. □

A.3 Averaged Unary Perturbations

A.3.1 Lower Bounds on the Partition Function

In the main text we stated the following two lower bounds on the log partition function $\ln Z$.

Proposition 5. Let $\alpha \in (-1, 0) \cup (0, \infty)$. For any subset $S \subseteq \{1, \ldots, n\}$ of the variables $x_1, \ldots, x_n$ we have $\ln Z \geq$

$$c + \frac{\ln \Gamma(1+\alpha)}{\alpha} - \frac{1}{\alpha} \ln \mathbb{E}\left[e^{-\alpha \max_{\mathbf{x}} \{\phi(\mathbf{x}) + \gamma_S(\mathbf{x}_S)\}}\right],$$

where $\mathbf{x}_S := \{x_i : i \in S\}$ and $\gamma_S(\mathbf{x}_S) \sim \text{Gumbel}(-c)$ independently for each setting of $\mathbf{x}_S$.

Proof. Let $\bar{S} := \{1, \ldots, n\} \setminus S$. First we handle the case $\alpha > 0$. We have trivially that

$$\text{pow}_{-\alpha} Z = \text{pow}_{-\alpha} \sum_{\mathbf{x}_S} \sum_{\mathbf{x}_{\bar{S}}} e^{\phi(\mathbf{x}_S, \mathbf{x}_{\bar{S}})} \leq \text{pow}_{-\alpha} \sum_{\mathbf{x}_S} \max_{\mathbf{x}_{\bar{S}}} e^{\phi(\mathbf{x}_S, \mathbf{x}_{\bar{S}})}.$$

The Weibull trick tells us that $\text{pow}_{-\alpha} \sum_y \text{pow}_{-1/\alpha} h(y) = \mathbb{E}_W[\min_y \frac{h(y)}{\Gamma(1+\alpha)} W(y)]$ where $\{W(y)\}_y \overset{iid}{\sim} \text{Weibull}(1, \alpha^{-1})$. Applying this to the summation over $\mathbf{x}_S$ on the right-hand side of the above inequality, we obtain

$$\text{pow}_{-\alpha} Z \leq \mathbb{E}_W \left[\min_{\mathbf{x}_S} \frac{\text{pow}_{-\alpha} \max_{\mathbf{x}_{\bar{S}}} e^{\phi(\mathbf{x}_S, \mathbf{x}_{\bar{S}})}}{\Gamma(1+\alpha)} W(\mathbf{x}_S)\right].$$

Expressing the Weibull random variable $W(\mathbf{x}_S)$ as $e^{-\alpha(\gamma_S(\mathbf{x}_S)+c)}$ with $\gamma_S(\mathbf{x}_S) \sim \text{Gumbel}(-c)$, the right-hand side can be simplified as follows:

$$\begin{aligned}
\text{pow}_{-\alpha} Z &\leq \frac{1}{\Gamma(1+\alpha)} \mathbb{E}_\gamma \left[\text{pow}_{-\alpha} \max_{\mathbf{x}_S} \max_{\mathbf{x}_{\bar{S}}} e^{\phi(\mathbf{x}_S, \mathbf{x}_{\bar{S}})} e^{\gamma_S(\mathbf{x}_S)+c}\right] \\
&= \frac{e^{-\alpha c}}{\Gamma(1+\alpha)} \mathbb{E}_\gamma \left[\exp\left(-\alpha \max_{\mathbf{x}} \{\phi(\mathbf{x}) + \gamma_S(\mathbf{x}_S)\}\right)\right].
\end{aligned}$$

Taking the logarithm and dividing by $-\alpha < 0$ yields the claimed result for positive α. For $\alpha \in (-1, 0)$ we proceed similarly, obtaining that

$$\begin{aligned}\operatorname{pow}_{-\alpha} Z &\geq \operatorname{pow}_{-\alpha} \sum_{\mathbf{x}_S} \max_{\mathbf{x}_{\bar{S}}} e^{\phi(\mathbf{x}_S, \mathbf{x}_{\bar{S}})} \\ &= \mathbb{E}_F \left[\min_{\mathbf{x}_S} \frac{\operatorname{pow}_{-\alpha} \max_{\mathbf{x}_{\bar{S}}} e^{\phi(\mathbf{x}_S, \mathbf{x}_{\bar{S}})}}{\Gamma(1+\alpha)} F(\mathbf{x}_S) \right],\end{aligned}$$

where $F(\mathbf{x}(S)) \sim \text{Fréchet}(1, -\alpha^{-1})$. Representing these random variables as $e^{-\alpha(\gamma_S(\mathbf{x}_S)+c)}$ with $\gamma_S(\mathbf{x}_S) \sim \text{Gumbel}(-c)$, simplifying as in the previous case and finally dividing the inequality by $-\alpha > 0$ yields the claimed result for $\alpha \in (-1, 0)$. □

Corollary 8. For any $\alpha \in (-1, 0) \cup (0, \infty)$, we have the lower bound $\ln Z \geq \mathcal{L}(\alpha)$, where

$$\mathcal{L}(\alpha) := c + \frac{\ln \Gamma(1+\alpha)}{\alpha} - \frac{1}{n\alpha} \ln \mathbb{E}\left[\exp\left(-n\alpha L\right)\right],$$

Proof. Applying Proposition 5 n times with all singleton sets $S = \{i\}$ and averaging the obtained lower bounds yields

$$\begin{aligned}\ln Z &\geq c + \frac{\ln \Gamma(1+\alpha)}{\alpha} - \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\alpha} \ln \mathbb{E}\left[\exp\left(-\alpha \max_{\mathbf{x}} \{\phi(\mathbf{x}) + \gamma_i(x_i)\}\right)\right] \\ &= c + \frac{\ln \Gamma(1+\alpha)}{\alpha} - \frac{1}{n\alpha} \ln \mathbb{E}\left[\exp\left(-\sum_{i=1}^{n} \alpha \max_{\mathbf{x}} \{\phi(\mathbf{x}) + \gamma_i(x_i)\}\right)\right] \\ &= c + \frac{\ln \Gamma(1+\alpha)}{\alpha} - \frac{1}{n\alpha} \ln \mathbb{E}\left[\exp\left(-n\alpha \frac{1}{n} \sum_{i=1}^{n} \max_{\mathbf{x}} \{\phi(\mathbf{x}) + \gamma_i(x_i)\}\right)\right],\end{aligned}$$

where the first equality used the fact that the perturbations $\gamma_i(x_i)$ are mutually independent for different indices i to replace the product of expectations with the expectation of the product. The claimed result follows by applying Jensen's inequality to swap the summation and the convex $\max_{\mathbf{x}}$ function, noting that the inequality works out the right way for both positive and negative α. □

Jensen's inequality can be used to relate the general lower bound $\mathcal{L}(\alpha)$ to the Gumbel trick lower bound $\mathcal{L}(0)$, showing that the former cannot be arbitrarily worse than the latter:

Proposition 15. *For all $\alpha \in (-1, 0)$, the lower bound $\mathcal{L}(\alpha)$ on $\ln Z$ satisfies*

$$\mathcal{L}(\alpha) \geq \mathcal{L}(0) + \frac{\ln \Gamma(1+\alpha)}{\alpha} + c.$$

Proof. Apply Jensen's inequality with the convex function $x \mapsto e^{-n\alpha}$ to the last term in the definition of $\mathcal{L}(\alpha)$, noting that the inequality works out the stated way for $\alpha < 0$. □

Note that $\frac{\ln \Gamma(1+\alpha)}{\alpha} + c \leq 0$ for $\alpha \in (-1, 0)$ so this result does *not* imply that the Fréchet lower bounds are tighter than the Gumbel lower bound $\mathcal{L}(0)$; it merely says that they cannot be arbitrarily worse than $\mathcal{L}(0)$.

A.3.2 Relationship Between Errors of Averaged-unary Gumbel Perturbations

In this section we write $\mathbf{x}^*$ for the (random) MAP configuration after average-unary perturbation of the potential function, i.e.,

$$\mathbf{x}^* := \operatorname*{argmax}_{\mathbf{x} \in \mathcal{X}} \left\{ \phi(\mathbf{x}) + \frac{1}{n} \sum_{i=1}^{n} \gamma_i(x_i) \right\}.$$

where $\{\gamma_i(x_i) \mid x_i \in \mathcal{X}_i, 1 \leq i \leq n\} \overset{\text{iid}}{\sim} \text{Gumbel}(-c)$. Let $q_{\text{avg}}(\mathbf{x}) := \mathbb{P}[\mathbf{x} = \mathbf{x}^*]$ be the probability mass function of $\mathbf{x}^*$. The Gumbel trick lower bound on the log partition function $\ln Z$ due to Hazan et al. (2013) is:

$$\ln Z \geq \mathcal{L}(0) = \mathcal{L}_\phi(0) := \mathbb{E}_\gamma \left[\min_{\mathbf{x} \in \mathcal{X}} \left\{ \phi(\mathbf{x}) + \frac{1}{n} \sum_{i=1}^{n} \gamma_i(x_i) \right\} \right]. \tag{A.1}$$

We show that the gap of this Gumbel trick lower bound on $\ln Z$ upper bounds the KL divergence between the approximate distribution q_{avg} and the Gibbs distribution p. To this end, we first need an entropy bound for q_{avg} analogous to Theorem 1 of (Maji et al., 2014).

Theorem 12. *The entropy of q_{avg} can be lower bounded using expected values of max-perturbations as follows:*

$$H(q_{avg}) \geq \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\gamma_i} \left[\gamma_i(x_i^*) \right].$$

Remark. Theorem 1 of (Maji et al., 2014) and this Theorem 12 differ in three aspects: (1) the former is an upper bound and the latter is a lower bound, (2) the former sums the expectations while the latter averages them, and (3) the distributions q_{sum} and q_{avg} of $\mathbf{x}^*$ in the two theorems are different.

Proof. By the duality relation between negative entropy and the log partition function (Wainwright and Jordan, 2008), the entropy $H(q_{\text{avg}})$ of the unary-avg perturb-max distribution q_{avg} can be expressed as

$$H(q_{\text{avg}}) = \inf_{\varphi} \left\{ \ln Z_{\varphi} - \sum_{\mathbf{x} \in \mathcal{X}} q_{\text{avg}}(\mathbf{x}) \varphi(\mathbf{x}) \right\},$$

where the variable φ ranges over all potential functions on $\mathcal{X}$, and $Z_{\varphi} = \sum_{\mathbf{x} \in \mathcal{X}} \exp \varphi(\mathbf{x})$. Applying the Gumbel trick lower bound on the log partition function gives

$$H(q_{\text{avg}}) \geq \inf_{\varphi} \left\{ \mathcal{L}_{\varphi}(0) - \sum_{\mathbf{x} \in \mathcal{X}} q_{\text{avg}}(\mathbf{x}) \varphi(\mathbf{x}) \right\},$$

Proposition 16 in Appendix A.4 shows that $\mathcal{L}_{\varphi}(0)$ is a convex function of φ. The expression $-\sum_{\mathbf{x} \in \mathcal{X}} q(\mathbf{x}) \varphi(\mathbf{x})$ is a linear function of φ, so also convex, and thus as a sum of two convex functions, the quantity $\mathcal{L}_{\varphi}(0) - \sum_{\mathbf{x} \in \mathcal{X}} q(\mathbf{x}) \varphi(\mathbf{x})$ within the infimum is a convex function of φ. Moreover, Proposition 17 in Appendix A.4 tells us that the partial derivatives can be computed as

$$\frac{\partial}{\partial \varphi(\mathbf{x})} \left(\mathcal{L}_{\varphi}(0) - \sum_{\mathbf{x} \in \mathcal{X}} q_{\text{avg}}(\mathbf{x}) \varphi(\mathbf{x}) \right) = q_{\varphi}(\mathbf{x}) - q_{\text{avg}}(\mathbf{x}),$$

where $q_{\varphi}(\mathbf{x})$ is the unary-avg perturb-max distribution associated with the potential function φ. Proposition 18 in Appendix A.4 confirms that these partial derivatives are continuous, so we observe that as a function of φ, the expression $\mathcal{L}_{\varphi}(0) - \sum_{\mathbf{x} \in \mathcal{X}} q_{\text{avg}}(\mathbf{x}) \varphi(\mathbf{x})$ is a convex function with continuous partial derivatives, so it is a differentiable convex function. This is sufficient to establish that the point $\varphi = \phi$ is a global minimum of this function (Wright and Nocedal, 1999). Hence

$$\begin{aligned}
H(q_{\text{avg}}) &\geq \inf_{\varphi} \left\{ \mathcal{L}_{\varphi}(0) - \sum_{\mathbf{x} \in \mathcal{X}} q_{\text{avg}}(\mathbf{x}) \varphi(\mathbf{x}) \right\} \\
&= \mathcal{L}_{\phi}(0) - \sum_{\mathbf{x} \in \mathcal{X}} q_{\text{avg}}(\mathbf{x}) \phi(\mathbf{x}) \\
&= \sum_{\mathbf{x} \in \mathcal{X}} q_{\text{avg}}(\mathbf{x}) \mathbb{E}_{\gamma} \left[\phi(\mathbf{x}) + \frac{1}{n} \sum_{i=1}^{n} \gamma_i(x_i) \mid \mathbf{x} = \mathbf{x}^* \right] - \sum_{\mathbf{x} \in \mathcal{X}} q_{\text{avg}}(\mathbf{x}) \phi(\mathbf{x}) \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\gamma_i} \left[\gamma_i(x_i^*) \right],
\end{aligned}$$

where we conditioned on the maximizing configuration $\mathbf{x}^*$ when expanding $\mathcal{L}_{\phi}(0)$. $\square$

Remark. This proof proceeded in the same way as the proof of Maji et al. (2014) for the upper bound, except that establishing the minimizing configuration of the infimum is a non-trivial step that is actually required in this case. The second revision of (Hazan et al., 2019) computes the derivative of $\mathcal{U}_\varphi(0) - \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{sum}}(\mathbf{x})\varphi(\mathbf{x})$, which is similar to our $\mathcal{L}_\varphi(0) - \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{avg}}(\mathbf{x})\varphi(\mathbf{x})$, by differentiating under the expectation.

Equipped with Theorem 12, we can now show a link between the approximation "errors" of the averaged-unary perturbation MAP configuration distribution q_{avg} (to the Gibbs distribution p) and estimate $\mathcal{L}(0)$ (to $\ln Z$).

Proposition 7. Let p be the Gibbs distribution on $\mathcal{X}$. Then

$$\underbrace{\ln Z - \mathcal{L}(0)}_{\text{error in } \ln Z \text{ bound}} \geq \underbrace{\mathrm{KL}(q_{\text{avg}} \parallel p)}_{\text{sampling error}} \geq 0.$$

Remark. While we knew from Hazan et al. (2013) that $\ln Z - \mathcal{L}(0) \geq 0$ (i.e. that $\mathcal{L}(0)$ is a lower bound on $\ln Z$), this is a stronger result showing that the size of the gap is an upper bound on the KL divergence between the average-unary perturbation MAP distribution q_{avg} and the Gibbs distribution p.

Proof. The Kullback-Leibler divergence in question expands as

$$\begin{aligned}\mathrm{KL}(q_{\text{avg}} \parallel p) &= -H(q_{\text{avg}}) - \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{avg}}(\mathbf{x}) \ln \frac{\exp \phi(\mathbf{x})}{\sum_{\tilde{\mathbf{x}}\in\mathcal{X}} \exp \phi(\tilde{\mathbf{x}})} \\ &= -H(q_{\text{avg}}) - \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{avg}}(\mathbf{x})\phi(\mathbf{x}) + \ln Z.\end{aligned}$$

From the proof of Theorem 12 we know that $H(q_{\text{avg}}) \geq \mathcal{L}(0) - \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{avg}}(\mathbf{x})\phi(\mathbf{x})$, so

$$\mathrm{KL}(q_{\text{avg}} \parallel p) \leq -\mathcal{L}(0) + \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{avg}}(\mathbf{x})\phi(\mathbf{x}) - \sum_{\mathbf{x}\in\mathcal{X}} q_{\text{avg}}(\mathbf{x})\phi(\mathbf{x}) + \ln Z = \ln Z - \mathcal{L}(0). \square$$

A.4 Technical Results

In this section we write $\mathcal{L}(\phi)$ instead of $\mathcal{L}_\phi(0)$ for the Gumbel trick lower bound on $\ln Z$ associated with the potential function ϕ, see equation (A.1).

Proposition 16. *The Gumbel trick lower bound $\mathcal{L}(\phi)$, viewed as a function of the potentials ϕ, is convex.*

Proof. Convexity can be proved directly from definition. Let ϕ_1 and ϕ_2 be two arbitrary potential functions on a discrete product space $\mathcal{X}$, and let $\lambda \in [0, 1]$. Then

$$\begin{aligned}
&\mathcal{L}(\lambda\phi_1 + (1-\lambda)\phi_2) \\
&= \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \lambda\phi_1(\mathbf{x}) + (1-\lambda)\phi_2(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i) \right\} \right] \\
&= \mathbb{E}_\gamma \left[\max_{\mathbf{x}\in\mathcal{X}} \left\{ \lambda \left(\phi_1(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i) \right) + (1-\lambda) \left(\phi_2(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i) \right) \right\} \right] \\
&\leq \mathbb{E}_\gamma \left[\lambda \max_{\mathbf{x}\in\mathcal{X}} \left\{ \phi_1(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i) \right\} + (1-\lambda) \max_{\mathbf{x}\in\mathcal{X}} \left\{ \phi_2(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i) \right\} \right] \\
&= \lambda\mathcal{L}(\phi_1) + (1-\lambda)\mathcal{L}(\phi_2),
\end{aligned}$$

where we have used convexity of the max function to obtain the inequality, and linearity of expectation to arrive at the final equality. □

Remark. This convexity proof goes through for other (low-dimensional) perturbations as well, e.g. it also works for $\mathcal{U}_\phi(0)$.

Proposition 17. *The Gumbel trick lower bound $\mathcal{L}(\phi)$, viewed as a function of the potentials ϕ, has partial derivatives*

$$\frac{\partial}{\partial\phi(\tilde{\mathbf{x}})}\mathcal{L}(\phi) = q_\phi(\tilde{\mathbf{x}}),$$

where q_ϕ is the probability mass function of the average-unary perturbation MAP configuration's distribution associated with the potential function ϕ.

Proof. Let $\tilde{\mathbf{x}} \in \mathcal{X}$, so that $\phi(\tilde{\mathbf{x}})$ is a general component of ϕ, and let $e_{\tilde{\mathbf{x}}}$ be the indicator vector of $\tilde{\mathbf{x}}$. For any $\delta \in \mathbb{R}$, the change in the lower bound $\mathcal{L}$ due to replacing $\phi(\tilde{\mathbf{x}})$

with $\phi(\tilde{\mathbf{x}}) + \delta$ is $\mathcal{L}(\phi + \delta e_{\tilde{\mathbf{x}}}) - \mathcal{L}(\phi) =$

$$\begin{aligned}
&\mathbb{E}_\gamma\left[\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \delta\mathbb{1}\{\mathbf{x}=\tilde{\mathbf{x}}\} + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i)\right\}\right] - \mathbb{E}_\gamma\left[\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i)\right\}\right] \\
&= \mathbb{E}_\gamma\left[\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \delta\mathbb{1}\{\mathbf{x}=\tilde{\mathbf{x}}\} + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i)\right\} - \max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i)\right\}\right] \\
&= \mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma)\right],
\end{aligned}$$

by linearity of expectation, where we have denoted by $\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma)$ the change in maximum due to replacing the potential $\phi(\tilde{\mathbf{x}})$ with $\phi(\tilde{\mathbf{x}}) + \delta$. Let's condition on the argmax before modifying ϕ:

$$\begin{aligned}
\mathcal{L}(\phi + \delta e_{\tilde{\mathbf{x}}}) - \mathcal{L}(\phi) &= \mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma)\right] \\
&= \sum_{\mathbf{x}\in\mathcal{X}} q_\phi(\mathbf{x})\mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma) \mid \mathbf{x} \text{ is the original argmax}\right].
\end{aligned}$$

Now let's condition on the size of the gap G between the maximum and the runner-up:

$$\begin{aligned}
&\mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma) \mid \mathbf{x} \text{ is the original argmax}\right] \\
&= \mathbb{P}(G \le |\delta|)\mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma) \mid \mathbf{x} \text{ is the original argmax}, G \le |\delta|\right] \\
&+ \mathbb{P}(G > |\delta|)\mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma) \mid \mathbf{x} \text{ is the original argmax}, G > |\delta|\right]
\end{aligned}$$

Let's examine all four terms on the right-hand side one by one:

1. $\mathbb{P}(G \le |\delta|) \to \mathbb{P}(G = 0) = 0$ as $\delta \to 0$ by monotonicity of measure.
2. $\mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma) \mid \mathbf{x} \text{ is the original argmax}, G \le |\delta|\right] \le \delta$ since $|\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma)| \le |\delta|$ always holds.
3. $\mathbb{P}(G > |\delta|) \to \mathbb{P}(G \ge 0) = 1$ as $\delta \to 0$ by monotonicity of measure.
4. $\mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma) \mid \mathbf{x} \text{ is the original argmax}, G > |\delta|\right] = \delta\mathbb{1}\{\mathbf{x} = \tilde{\mathbf{x}}\}$ since in this case both maximizations in the definition of $\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma)$ are maximized at $\mathbf{x}$.

Therefore, as $\delta \to 0$,

$$\mathbb{E}_\gamma\left[\Delta(\phi, \delta, \tilde{\mathbf{x}}, \gamma) \mid \mathbf{x} \text{ is the original argmax}\right] = o(1)o(\delta) + (1 + o(1))\delta\mathbb{1}\{\mathbf{x} = \tilde{\mathbf{x}}\}$$

Putting things together, we have

$$\begin{aligned}\lim_{\delta\to 0}\frac{\mathcal{L}(\phi+\delta e_{\tilde{\mathbf{x}}})-\mathcal{L}(\phi)}{\delta} &= \sum_{\mathbf{x}\in\mathcal{X}} q_\phi(\mathbf{x}) \lim_{\delta\to 0}\frac{1}{\delta}\mathbb{E}_\gamma\left[\Delta(\phi,\delta,\tilde{\mathbf{x}},\gamma) \mid \mathbf{x} \text{ is the original argmax}\right] \\ &= \sum_{\mathbf{x}\in\mathcal{X}} q_\phi(\mathbf{x})\mathbb{1}\{\mathbf{x}=\tilde{\mathbf{x}}\} \\ &= q_\phi(\tilde{\mathbf{x}}),\end{aligned}$$

which proves the stated claim directly from definition of a partial derivative. □

Proposition 18. *The probability mass function q_ϕ of the average-unary perturbation MAP configuration's distribution associated with a potential function ϕ is continuous in ϕ.*

Proof. For any $\mathbf{x}^* \in \mathcal{X}$ we have from definition

$$\begin{aligned}q_\phi(\mathbf{x}^*) &= \mathbb{P}\left[\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i)\right\}\right] \\ &= \mathbb{P}\left[\phi(\mathbf{x}^*) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i^*) > \max_{\mathbf{x}\in\mathcal{X}\setminus\{\mathbf{x}^*\}}\left\{\phi(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i)\right\}\right] \\ &= \mathbb{E}\left[\mathbb{1}\left\{\phi(\mathbf{x}^*) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i^*) > \max_{\mathbf{x}\in\mathcal{X}\setminus\{\mathbf{x}^*\}}\left\{\phi(\mathbf{x}) + \frac{1}{n}\sum_{i=1}^n \gamma_i(x_i)\right\}\right\}\right]\end{aligned}$$

which is continuous in ϕ by continuity of max, of $\mathbb{1}\{\cdot > \cdot\}$ (as a function of ϕ) and by the Bounded Convergence Theorem. □

Remark. The results above show that the Gumbel trick lower bound $\mathcal{L}(\phi)$, viewed as a function of the potentials ϕ, is convex and has continuous partial derivatives.

Appendix B

Rényi Entropy Estimation

Here we provide proofs and details that were omitted in the main text of Chapter 3.

Proposition 8. For $A \subseteq \mathcal{X}$, let $\hat{H}_\alpha^A(p)$ be the estimator of $H_\alpha(p)$ that shares the noise among configurations in A, and uses independent noise for the others:

$$\hat{H}_\alpha^A(p) := \frac{\alpha}{1-\alpha}\frac{1}{M}\sum_{m=1}^{M}\left[\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\frac{1}{\alpha}\gamma_0^{(m)}(\mathbf{x})\right\}-\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\gamma_1^{(m)}(\mathbf{x})\right\}\right] \tag{B.1}$$

where $\gamma_0^{(m)}(\mathbf{x}) = \gamma_1^{(m)}(\mathbf{x})$ for all $\mathbf{x} \in A$, and otherwise all γ's are i.i.d. Gumbel$(-c)$. Then

$$\operatorname{var}\left[\hat{H}_\alpha^{\mathcal{X}}(p)\right] \leq \operatorname{var}\left[\hat{H}_\alpha^{\emptyset}(p)\right] = \frac{1}{M}\frac{1+\alpha^2}{(1-\alpha)^2}\frac{\pi^2}{6} \tag{B.2}$$

Proof. As perturbations across different samples $m = 1, \ldots, M$ are independent, and the variance of a sum of independent random variables is the sum of individual variances, it suffices to consider the case $M = 1$ and focus on

$$\begin{aligned}
&\operatorname{var}\left(\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\frac{1}{\alpha}\gamma_0(\mathbf{x})\right\}-\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\gamma_1(\mathbf{x})\right\}\right)\\
&=\operatorname{var}\left(\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\frac{1}{\alpha}\gamma_0(\mathbf{x})\right\}\right)+\operatorname{var}\left(\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\gamma_1(\mathbf{x})\right\}\right)\\
&\quad-2\operatorname{Cov}\left(\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\frac{1}{\alpha}\gamma_0(\mathbf{x})\right\},\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\gamma_1(\mathbf{x})\right\}\right)\\
&=\frac{1}{\alpha^2}\frac{\pi^2}{6}+\frac{\pi^2}{6}-2\operatorname{Cov}\left(\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\frac{1}{\alpha}\gamma_0(\mathbf{x})\right\},\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x})+\gamma_1(\mathbf{x})\right\}\right),
\end{aligned} \tag{B.3}$$

where the variances were evaluated by recognizing Gumbel distributions as distributions of max-perturbations. When $A = \emptyset$ and γ_0, γ_1 are mutually independent, the covariance vanishes and $\operatorname{var}\left[\hat{H}_\alpha^{\emptyset}(p)\right]$ evaluates to the expression on the right-hand

side of Equation 3.14 in the proposition statement, as required (upon adding back the appropriate $\frac{1}{M}$ scaling). Otherwise, the functions $f, g : \mathbb{R}^{\mathcal{X}} \to \mathbb{R}$ defined respectively by $f(\mathbf{z}) := \max_{\mathbf{x}\in\mathcal{X}} \left\{\phi(\mathbf{x}) + \frac{1}{\alpha} z_{\mathbf{x}}\right\}$ and $g(\mathbf{z}) := \max_{\mathbf{x}\in\mathcal{X}} \{\phi(\mathbf{x}) + z_{\mathbf{x}}\}$ are both non-decreasing in all of their arguments $z_{\mathbf{x}} \in \mathbb{R}$, so the product measure version of Harris' inequality (a special case of the FKG inequality) applies to conclude that the covariance is non-negative. It follows that $\operatorname{var}\left[\hat{H}_\alpha^{\mathcal{X}}(p)\right] \leq \operatorname{var}\left[\hat{H}_\alpha^{\emptyset}(p)\right]$. □

Proposition 19. *Suppose the normalizer Z of the distribution p is known, and let $\hat{H}_\alpha(p)$ be the estimator of $H_\alpha(p)$ for normalized distributions based on Equation 3.8 (i.e. without a control variate). When p is a distribution uniformly distributed on its support then $\hat{H}_\alpha^{\mathcal{X}}(p)$ is a lower-variance estimator than $\hat{H}_\alpha(p)$ if and only if $\alpha < 2$.*

Proof. Recall that the control variate estimator has lower variance if and only if the variance of the control variate is smaller than twice the covariance between the control variate and the original estimator.

Write $\mathcal{X} = \mathcal{X}_0 \sqcup \mathcal{X}_1$ such that $\phi(\mathbf{x}) = -\infty$ for $\mathbf{x} \in \mathcal{X}_0$ and $\phi(\mathbf{x}) = a := -\ln|\mathcal{X}_1|$ for $\mathbf{x} \in \mathcal{X}_1$. Then

$$\begin{aligned}
&2\operatorname{Cov}\left(\max_{\mathbf{x}\in\mathcal{X}}\left\{\phi(\mathbf{x}) + \frac{1}{\alpha}\gamma(\mathbf{x})\right\}, \max_{\mathbf{x}\in\mathcal{X}}\{\phi(\mathbf{x}) + \gamma(\mathbf{x})\}\right) \\
&= 2\operatorname{Cov}\left(\max_{\mathbf{x}\in\mathcal{X}_1}\left\{a + \frac{1}{\alpha}\gamma(\mathbf{x})\right\}, \max_{\mathbf{x}\in\mathcal{X}_1}\{a + \gamma(\mathbf{x})\}\right) \\
&= 2\frac{1}{\alpha}\operatorname{var}\left(\max_{\mathbf{x}\in\mathcal{X}_1}\gamma(\mathbf{x})\right) = \frac{2}{\alpha}\frac{\pi^2}{6}.
\end{aligned}$$

This is larger than the $\frac{\pi^2}{6}$ variance of the control variate if and only if $\alpha < 2$. □

Appendix C

Differentially Private Database Release

C.1 Proofs

Here we provide proofs for the results stated in Chapter 4, together with additional supporting lemmas required for these proofs.

C.1.1 Synthetic Data Subspace Algorithm: Consistency

Before proving Theorem 2, we obtain a Lemma showing that the "projection error" incurred due to projecting the KME $\hat{\mu}_X$ onto the finite-dimensional subspace $\mathcal{H}_M$ spanned by the synthetic data points, quantified by the RKHS distance between $\hat{\mu}_X$ and the projection $\overline{\mu}_X$, converges to 0 as $N \to \infty$:

Lemma 4. *Let $\mathcal{X}$ be a compact metric space and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a continuous kernel on $\mathcal{X}$. Suppose that the synthetic data points $z_1, z_2, \ldots$ are sampled i.i.d. from a probability distribution q on $\mathcal{X}$. If the support $\mathrm{supp}(X)$ of X is included in the support of q, then*

$$\|\overline{\mu}_X - \hat{\mu}_X\|_{\mathcal{H}} \xrightarrow{\mathbb{P}} 0 \text{ as } N \to \infty. \tag{C.1}$$

Proof. Let $\varepsilon > 0$. As k is continuous on $\mathcal{X} \times \mathcal{X}$, which as a product of compact spaces is itself compact by Tychonoff's theorem, the kernel k is uniformly continuous and in particular there exists $\delta > 0$ such that for all $x, x' \in \mathcal{X}$ we have $|k(x,x) - k(x,x')| < \varepsilon^2/2$ whenever $\|x - x'\|_{\mathcal{X}} < \delta$. As $\mathcal{X}$ is compact, it is totally bounded, and thus so is its subset $\mathrm{supp}(X)$. Therefore $\mathrm{supp}(X)$ can be covered with finitely many open balls

$B_1, \ldots, B_K$ of radius $\delta/2$. Let the sequence $z_1, z_2, \ldots$ be sampled i.i.d. from q, and let E_M be the event that at least ones of these K balls contains no element of $z_1, \ldots, z_M$. Since $\mathrm{supp}(X) \subseteq \mathrm{supp}(q)$ by assumption, we have $q(B_k) > 0$ for all $k = 1, \ldots, K$ and therefore $\mathbb{P}[E_M] \to 0$ as $M \to \infty$.

Note that if all K balls contain an element of $z_1, \ldots, z_M$ (i.e., E_M^C holds), then for each $x \in \mathrm{supp}(X)$ one can find $1 \leq m(x) \leq M$ such that $\|x - z_{m(x)}\| < \delta/2 + \delta/2 = \delta$. In that case

$$\begin{aligned}
\|\overline{\mu}_X - \hat{\mu}_X\|_{\mathcal{H}} &= \inf_{h \in \mathcal{H}_M} \|h - \hat{\mu}_X\|_{\mathcal{H}} && \text{[property of projection]} \\
&\leq \left\| \frac{1}{N} \sum_{n=1}^{N} k(z_{m(x_n)}, \cdot) - \hat{\mu}_X \right\|_{\mathcal{H}} && \text{[as } \tfrac{1}{N} \textstyle\sum_{n=1}^{N} k(z_{m(x_n)}, \cdot) \in \mathcal{H}_M\text{]} \\
&\leq \frac{1}{N} \sum_{n=1}^{N} \left\| k(z_{m(x_n)}, \cdot) - k(x_n, \cdot) \right\|_{\mathcal{H}} && \text{[Triangle inequality]} \\
&< \frac{1}{N} \sum_{n=1}^{N} \varepsilon && \text{[see below]} \\
&= \varepsilon, && \text{(C.2)}
\end{aligned}$$

where we have used the reproducing property, the Triangle inequality and our choices of δ and $z_{m(x_n)}$ to see that for all $1 \leq n \leq N$,

$$\begin{aligned}
&\left\| k(z_{m(x_n)}, \cdot) - k(x_n, \cdot) \right\|_{\mathcal{H}} \\
&= \langle k(z_{m(x_n),\cdot} - k(x_n, \cdot), k(z_{m(x_n),\cdot} - k(x_n, \cdot) \rangle_{\mathcal{H}}^{1/2} \\
&= \Big(k(z_{m(x_n)}, z_{m(x_n)}) - 2k(z_{m(x_n)}, x_n) + k(x_n, x_n) \Big)^{1/2} \\
&\leq \Big(|k(z_{m(x_n)}, z_{m(x_n)}) - k(z_{m(x_n)}, x_n)| + |k(x_n, x_n) - k(z_{m(x_n)}, x_n)| \Big)^{1/2} \\
&< \left(\frac{\varepsilon^2}{2} + \frac{\varepsilon^2}{2} \right)^{1/2} = \varepsilon. && \text{(C.3)}
\end{aligned}$$

Hence we have that $\mathbb{P}\left[\|\overline{\mu}_X - \hat{\mu}_X\|_{\mathcal{H}} > \varepsilon \right] \leq \mathbb{P}[E_M] \to 0$ as $M \to \infty$. But since $\varepsilon > 0$ was arbitrary and $M \to \infty$ as $N \to \infty$ by construction, the claimed convergence in probability result follows from definition. □

Theorem 10. Let $\mathcal{X}$ be a compact metric space and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a continuous kernel on $\mathcal{X}$. Suppose that the synthetic data points $z_1, z_2, \ldots$ are sampled i.i.d. from a probability distribution q on $\mathcal{X}$. If the support of X is included in the support of q, then Algorithm 2 outputs a consistent estimator of the kernel mean embedding μ_X in

the sense that

$$\sum_{m=1}^{M} w_m k(z_m, \cdot) \xrightarrow{\mathbb{P}} \mu_X \qquad \text{as } N \to \infty. \tag{C.4}$$

Proof. Using the Triangle inequality, we can upper bound the RKHS distance between the output $\tilde{\mu}_X$ of Algorithm 2 and the true kernel mean embedding μ_X as follows:

$$\|\tilde{\mu}_X - \mu_X\|_{\mathcal{H}} \leq \underbrace{\|\tilde{\mu}_X - \overline{\mu}_X\|_{\mathcal{H}}}_{\text{privacy error}} + \underbrace{\|\overline{\mu}_X - \hat{\mu}_X\|_{\mathcal{H}}}_{\text{projection error}} + \underbrace{\|\hat{\mu}_X - \mu_X\|_{\mathcal{H}}}_{\text{finite sample error}}. \tag{C.5}$$

The finite sample error tends to 0 as $N \to \infty$ by the law of large numbers, while the projection error tends to 0 as $N \to \infty$ by Lemma 4. For the privacy error, using orthonormality of the basis $\{b_1, \ldots, b_F\}$ we have

$$\|\tilde{\mu}_X - \overline{\mu}_X\|_{\mathcal{H}}^2 = \left\|\sum_{f=1}^{F}(\beta_f - \alpha_f)b_f\right\|_{\mathcal{H}}^2 = \sum_{f=1}^{F}(\beta_f - \alpha_f)^2 = \frac{8\ln\frac{1.25}{\delta}}{N^2\varepsilon^2} F \frac{1}{F}\sum_{f=1}^{F}\mathcal{N}(0,1)^2. \tag{C.6}$$

As a function of N, the size of the basis $F \in \mathbb{N}$ is a non-decreasing function, so it either converges to some $L \in \mathbb{N}$, in which case the obtained expression clearly tends to 0 as $N \to \infty$ with probability 1, or $F \to \infty$ as $N \to \infty$. In this latter case $\frac{1}{F}\sum_{f=1}^{F}\mathcal{N}(0,1)^2 \to 1$ as $N \to \infty$ a.s. by the strong law of large numbers, and $F/N^2 \to 0$ as $N \to \infty$ since $F \leq M = o(N^2)$. Hence the privacy error goes to 0 as $N \to \infty$ either way, as required to complete the proof. □

Theorem 13. *Suppose that the kernel k is c_0-universal Sriperumbudur et al. (2011) and f is any continuous function mapping from $\mathcal{X}$ to some space $\mathcal{Y}$. Let $C \geq 1$ be any finite constant. If line 8 of Algorithm 2 is replaced with a regularised reduced set method solving the constrained minimisation problem*

$$\mathbf{w} = \operatorname*{argmin}_{\mathbf{u}:\|\mathbf{u}\|_1 \leq C} \left\|\tilde{\mu}_X - \sum_{m=1}^{M} u_m k(z_m, \cdot)\right\|_{\mathcal{H}}, \tag{C.7}$$

then the points output by Algorithm 2 yield a consistent estimator of the kernel mean embedding $\mathbb{E}[k(f(X), \cdot)]$ of $f(X)$ in the sense that

$$\sum_{m=1}^{M} w_m k(f(z_m), \cdot) \xrightarrow{\mathbb{P}} \mu_{f(X)} \qquad \text{as } N \to \infty. \tag{C.8}$$

Proof. Let $\mu_X^{\text{out}} := \sum_{m=1}^{M} w_m k(z_m, \cdot)$ be the RKHS element output by Algorithm 2 after adding the stated regularisation. First we show that despite the regularisation, μ_X^{out} remains a consistent estimator of the true kernel mean embedding μ_X as $N \to \infty$.

The modification introduces an additional regularisation error term $\|\mu_X^{\text{out}} - \tilde{\mu}_X\|_{\mathcal{H}}$ into the upper bound on $\|\mu_X^{\text{out}} - \mu_X\|$, compared to the corresponding bound (C.5) in the proof of Theorem 10. So to show the first desired consistency result, it remains to show that this extra regularisation error term converges to 0 in probability as $N \to \infty$. To this end, let $\varepsilon > 0$ be arbitrary. Define $\delta > 0$, the sequence $z_1, z_2, \ldots$ and $m(x)$ for $x \in \mathcal{X}$ as in the proof of Lemma 4. Note that the RKHS element $\frac{1}{N}\sum_{n=1}^{N} k(z_{m(x_n)}, \cdot)$ is in the feasible set of the regularised minimisation problem (C.7), because the sum of absolute values of expansions coefficients is

$$\sum_{m=1}^{M} \sum_{n:m(x_n)=n} \frac{1}{M} = \sum_{n=1}^{N} \frac{1}{N} = 1 \leq C. \tag{C.9}$$

Therefore the regularisation error can be upper bounded as

$$\begin{aligned}
\|\mu_X^{\text{out}} - \tilde{\mu}_X\|_{\mathcal{H}} &\leq \left\| \frac{1}{N}\sum_{n=1}^{N} k(z_{m(x_n)}, \cdot) - \tilde{\mu}_X \right\|_{\mathcal{H}} && \text{[property of min]} \\
&\leq \|\tilde{\mu}_X - \hat{\mu}_X\|_{\mathcal{H}} + \left\| \hat{\mu}_X - \frac{1}{N}\sum_{n=1}^{N} k(z_{m(x_n)}, \cdot) \right\|_{\mathcal{H}}. && \text{[Triangle inequality]}
\end{aligned}$$

The first term goes to 0 as $N \to \infty$ by the argument given in the proof of Theorem 10. The probability that the second term is larger than ε converges to 0 as $N \to \infty$ using the argument given in the proof of Lemma 4. Hence we have the desired convergence of the modified Algorithm 2's output μ_X^{out} to the true kernel mean embedding μ_X as $N \to \infty$, in probability.

This means that the modified algorithm still outputs a consistent estimator of the kernel mean embedding of μ_X. Moreover, the weights in the released finite expansion now have their L_1 norm $\sum_{m=1}^{M} |w_m|$ bounded by the constant C by construction, so Theorem 1 of Simon-Gabriel et al. (2016) applies and gives the desired conclusion regarding consistency of the estimator for the kernel mean embedding $\mu_{f(X)}$ of $f(X)$. □

C.1.2 Synthetic Data Subspace Algorithm: Convergence Rates

Towards proving the convergence rate of Proposition 11, we will make use of the following Lemma 5, which is a refinement of the corresponding consistency result

of Lemma 4 above. It uses the Lipschitz assumption on the kernel to establish a quantitative dependence between ε and δ, and the condition on q to establish a dependence between δ, K and $\mathbb{P}[E_M]$.

Lemma 5. *Suppose that $\mathcal{X}$ is a bounded subset of $\mathbb{R}^D$, the kernel k is Lipschitz with some Lipschitz constant $L \in \mathbb{R}^+$, and the synthetic data points $z_1, z_2, \ldots$ are sampled i.i.d. from a distribution q whose density is bounded away from 0 on any bounded subset of $\mathbb{R}^D$. Then*

$$\begin{aligned} \forall \gamma \in (0,1), a > 0 \qquad \exists C \in \mathbb{R}, \varepsilon_0 > 0 \qquad \forall \varepsilon \in (0, \varepsilon_0) \\ M \geq C\varepsilon^{-2D-a} \quad \Rightarrow \quad \mathbb{P}\left[\|\hat{\mu}_X - \bar{\mu}_X\|_{\mathcal{H}} \geq \varepsilon\right] \leq \gamma. \end{aligned} \tag{C.10}$$

Proof. Let $\gamma \in (0,1)$ and $a > 0$. Suppose for the moment that C and ε_0 have already been chosen based on $\mathcal{X}, q, \gamma, a$ and based on the Lipschitz constant L of the kernel k. Let $\varepsilon \in (0, \varepsilon_0)$ and suppose that $M \geq C\varepsilon^{-2D-a}$.

Define $\delta = \frac{\varepsilon^2}{2L}$ and let $B_1, \ldots, B_K$ be a covering of $\mathrm{supp}(X)$ with K open balls of radii $\frac{\delta}{2}$. By the Lipschitz property

$$\|x - x'\|_{\mathcal{X}} < \delta \quad \Rightarrow \quad |k(x,x') - k(x,x)| \leq L\|x - x'\|_{\mathcal{X}} < L\delta = \frac{\varepsilon^2}{2}, \tag{C.11}$$

and so by the argument appearing in the proof of Lemma 4, if each ball B_k contains at least one synthetic data point z_m, then $\|\hat{\mu}_X - \bar{\mu}_X\|_{\mathcal{H}} < \varepsilon$. Therefore it suffices to show that if $M \geq C\varepsilon^{-2D(1+a)}$, then the probability of some of the balls not containing any synthetic data point is at most γ.

To this end, let us look at the number of balls K, and the probability that a synthetic data point lands in a particular ball, as functions of ε (via the ball radius $\frac{\delta}{2}$). First, since $\mathcal{X}$ is a bounded subset of $\mathbb{R}^D$, there exists $C_1 \in \mathbb{R}$ such that for all $\delta > 0$, the space $\mathcal{X}$ can be covered with $\lfloor C_1 \delta^{-D} \rfloor$ open balls of radii $\delta/2$. Second, since the density of q is assumed to be bounded away from 0 on any bounded subset of $\mathbb{R}^D$, there exists $C_2 \in \mathbb{R}$ such that $q(B_k) \geq C_2 \delta^D$ for all k.

Let A_k^M be the event that the ball B_k remains without a synthetic data point after M of them have been sampled. Then the probability of the event E_M that *any* of the

K balls remains empty can be upper bounded by a union bound as

$$\mathbb{P}[E_M] \leq \sum_{k=1}^{K} \mathbb{P}[A_k^M] = \sum_{k=1}^{K} (1 - q(B_k))^M \leq \sum_{k=1}^{K} (1 - C_2\delta^D)^M \leq K \exp\left(-MC_2\delta^D\right)$$
$$\leq C_1\delta^{-D} \exp\left(-MC_2\delta^D\right). \tag{C.12}$$

Solving for M, we can easily verify that $\mathbb{P}[E_M] \leq \gamma$ is ensured whenever

$$M \geq \frac{1}{C_2\delta^D}\left(D \ln \frac{1}{\delta} + \ln \frac{C_1}{\gamma}\right) = \frac{(2L)^D}{C_2}\frac{1}{\varepsilon^{2D}}\left(2 \ln \frac{1}{\varepsilon} + \ln \frac{C_1(2L)^D}{\gamma}\right). \tag{C.13}$$

Since $\ln \frac{1}{\varepsilon} < \frac{1}{\varepsilon^a}$ for all sufficiently small ε, we see that we could have chosen $\varepsilon_0 > 0$ and $C \in \mathbb{R}$ such that the right-hand side is at most $C\varepsilon^{-2D-a}$ for all $\varepsilon \in (0, \varepsilon_0)$. But the condition $M \geq C\varepsilon^{-2D-a}$ is satisfied by supposition, and so we conclude that $\mathbb{P}\left[\|\hat{\mu}_X - \bar{\mu}_X\|_{\mathcal{H}}\right] \leq \mathbb{P}[E_M] \leq \gamma$. □

Proposition 11 Suppose that $\mathcal{X}$ is a bounded subset of $\mathbb{R}^D$, the kernel k is Lipschitz, and the synthetic data points $z_1, z_2, \ldots$ are sampled i.i.d. from a distribution q whose density is bounded away from 0 on any bounded subset of $\mathbb{R}^D$. Then $M(N)$ can be chosen so that Algorithm 2 outputs an estimator that converges to the true kernel mean embedding μ_X in RKHS norm at a rate $\mathcal{O}_p(N^{-1/(D+1+c)})$, where c is any fixed positive number $c > 0$.

Proof. As in the proof of Theorem 10, we can decompose the error between the released element $\tilde{\mu}_X$ and the true μ_X as

$$\|\tilde{\mu}_X - \mu_X\|_{\mathcal{H}} \leq \underbrace{\|\hat{\mu}_X - \mu_X\|_{\mathcal{H}}}_{\text{finite sample error}} + \underbrace{\|\overline{\mu}_X - \hat{\mu}_X\|_{\mathcal{H}}}_{\text{projection error}} + \underbrace{\|\tilde{\mu}_X - \overline{\mu}_X\|_{\mathcal{H}}}_{\text{privacy error}}. \tag{C.14}$$

Using the standard empirical kernel mean embedding estimator, the finite sample error vanishes as $\mathcal{O}_p(N^{-1/2})$ (Muandet et al., 2016). From the proof of Theorem 10 we can see that the privacy error vanishes as $\mathcal{O}_p(\sqrt{F}/N) \subseteq \mathcal{O}_p(\sqrt{M}/N)$. Solving for ε in the statement of the preceding Lemma 5 we have that for all $\gamma \in (0, 1)$, $a > 0$ and all sufficiently large M,

$$\mathbb{P}\left[\|\hat{\mu}_X - \bar{\mu}_X\|_{\mathcal{H}} \geq \frac{1}{C}M^{-\frac{1}{2D+a}}\right] \leq \gamma. \tag{C.15}$$

The projection error thus vanishes at a rate $\mathcal{O}_p(M^{-1/(2D+a)})$, for any arbitrarily small $a > 0$. To achieve the claimed total rate $\mathcal{O}_p(N^{-1/(D+1+c)})$ we choose $M(N) = N^k$ with

$k = 1 - 4/(2D + a + 2)$, and verify that

$$\mathcal{O}_p\left(\frac{1}{\sqrt{N}} + M^{\frac{-1}{2D+a}} + \frac{\sqrt{M}}{N}\right) = \mathcal{O}_p\left(\frac{1}{\sqrt{N}} + N^{\frac{-k}{2D+a}} + \frac{\sqrt{N^k}}{N}\right) = \mathcal{O}_p\left(\frac{1}{\sqrt{N}} + N^{-\frac{1}{D+1+a/2}}\right)$$
$$= \mathcal{O}_p\left(N^{-\frac{1}{D+1+a/2}}\right), \tag{C.16}$$

and the claimed result follows by taking $a = 2c > 0$. □

Proposition 12 Suppose that a fixed proportion η of the private database can be published without modification. Using this part of the database as the synthetic data points, Algorithm 2 outputs a consistent estimator of μ_X that converges in RKHS norm at a rate $\mathcal{O}_p(N^{-1/2})$.

Proof. Let $\hat{\mu}^{\text{baseline}} := \frac{1}{M}\sum_{m=1}^{M} k(z_m, \cdot)$ be the baseline estimator that weights the M public points uniformly. Noting that $\hat{\mu}^{\text{baseline}} \in \mathcal{H}_M$ lies in the span of feature maps of synthetic data points, for the projection error as defined in equation (C.14) we have:

$$\begin{aligned}
\|\overline{\mu}_X - \hat{\mu}_X\|_{\mathcal{H}} &\leq \left\|\hat{\mu}^{\text{baseline}} - \hat{\mu}_X\right\|_{\mathcal{H}} && [\text{ property of projection }] \\
&= \left\|\hat{\mu}^{\text{baseline}} - \mu_X\right\|_{\mathcal{H}} + \|\hat{\mu}_X - \mu_X\|_{\mathcal{H}} && [\text{ Triangle inequality }] \\
&\in \mathcal{O}_p\left(M^{-1/2}\right) + \mathcal{O}_p\left(N^{-1/2}\right). && \text{(C.17)}
\end{aligned}$$

Using the error decomposition of equation (C.14) we thus have

$$\|\tilde{\mu}_X - \mu_X\|_{\mathcal{H}} \in \mathcal{O}_p\left(N^{-1/2} + (M^{-1/2} + N^{-1/2}) + \sqrt{M}/N\right), \tag{C.18}$$

and this is in $\mathcal{O}_p(N^{-1/2})$ when $M = \eta N$ is proportional to N. □

C.1.3 Synthetic Data Subspace Algorithm: Privacy

The proof of Proposition 10 rests on the following simple calculation:

Lemma 6. *If $k(x, x) \leq 1$ for all $x \in \mathcal{X}$, then the RKHS norm sensitivity of the empirical kernel mean embedding $\hat{\mu}_X$ with respect to changing one data point is at most $\frac{2}{N}$.*

Proof. Let $D = \{x_1, \ldots, x_N\}$ and $D' = \{x'_1, \ldots, x'_N\}$ be two databases of the same cardinality N, differing in a single row. Without loss of generality $x_n = x'_n$ for

$1 \leq n \leq N-1$. Let $\hat{\mu}_X$ and $\hat{\mu}'_X$ be the empirical kernel mean embeddings computed using D and D', respectively. Then

$$\begin{aligned}\|\hat{\mu}_X - \hat{\mu}'_X\|_{\mathcal{H}} &= \left\| \frac{1}{N}\sum_{n=1}^{N} k(x_n, \cdot) - \frac{1}{N}\sum_{n=1}^{N} k(x'_n, \cdot) \right\|_{\mathcal{H}} \\ &= \frac{1}{N}\|k(x_N, \cdot) - k(x'_N, \cdot)\|_{\mathcal{H}} \\ &\leq \frac{1}{N}\left(\|k(x_N, \cdot)\|_{\mathcal{H}} + \|k(x'_N, \cdot)\|_{\mathcal{H}}\right) \\ &= \frac{1}{N}\left(k(x_N, x_N)^{1/2} + k(x'_N, x'_N)^{1/2}\right) \\ &\leq \frac{2}{N}. \end{aligned} \tag{C.19}$$

As D and D' were arbitrary neighbouring databases, the claimed result follows. □

Proposition 10. If $k(x, x) \leq 1$ for all $x \in \mathcal{X}$, then Algorithm 2 is (ε, δ)-differentially private.

Proof. As the synthetic data points $z_1, \ldots, z_M$ do not depend on the private data, it suffices to show that the weights $w_1, \ldots, w_M$ are (ε, δ)-differentially private. However, these weights result from data-independent post-processing of the coefficients $\boldsymbol{\beta}$, which are a perturbed version of the coefficients $\boldsymbol{\alpha}$, with the perturbation provided by the privacy-protecting *Gaussian mechanism* Dwork and Roth (2014). It remains to verify that the Gaussian mechanism employs sufficiently scaled noise; in particular we need to verify that $2/N \geq \Delta_2 := \sup_{D,D':D\sim D'} \|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_2$.

But indeed, since $b_1, \ldots, b_F$ are orthonormal, for any $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ computed using neighbouring databases,

$$\|\boldsymbol{\alpha} - \boldsymbol{\alpha}'\|_2 = \left(\sum_{f=1}^{F}(\alpha_f - \alpha'_f)^2\right)^{1/2} = \left\|\overline{\hat{\mu}_N} - \overline{\hat{\mu}'_N}\right\|_{\mathcal{H}} \leq \|\hat{\mu}_N - \hat{\mu}'_N\|_{\mathcal{H}} \leq \frac{2}{N}, \tag{C.20}$$

(last inequality is Lemma 6) as required to verify the Gaussian mechanism. Then (ε, δ)-differential privacy for the entire algorithm follows. □

C.1.4 Random Features RKHS Algorithm: Consistency

As a preliminary lemma, we first show that a uniform convergence result for the random features ϕ translates into a bound on the error incurred by Algorithm 3 due to using random features instead of the original kernel k.

Lemma 7. *Let $\hat{\mu}_X^{out} := \sum_{m=1}^M w_m k(z_m, \cdot) \in \mathcal{H}$ be the element in $\mathcal{H}$ represented by the output of Algorithm 3. Let $\hat{\mu}_X^{\phi,out} := \sum_{m=1}^M w_m \phi(z_m)$ be the corresponding element in the random features RKHS $\mathcal{H}_\phi$. If the random feature scheme ϕ is such that $\sup_{x,x' \in \mathcal{X}} |\phi(x)^T \phi(x') - k(x, x')| < \delta$, then the following bound on the "random features error" holds:*

$$\left| \left\| \hat{\mu}_X^{\phi,out} - \hat{\mu}_X^{\phi} \right\|_{\mathcal{H}_\phi} - \left\| \hat{\mu}_X^{out} - \hat{\mu}_X \right\|_{\mathcal{H}} \right| \leq 2\sqrt{\delta}. \tag{C.21}$$

Proof. Expanding the RKHS norms using bilinearity of inner products, we have

$$\begin{aligned}
&\left| \left\| \hat{\mu}_X^{\phi,\mathrm{out}} - \hat{\mu}_X^{\phi} \right\|_{\mathcal{H}_\phi} - \left\| \hat{\mu}_X^{\mathrm{out}} - \hat{\mu}_X \right\|_{\mathcal{H}} \right| \\
&= \Bigg| \Bigg(\sum_{m_1=1}^M \sum_{m_2=1}^M w_{m_1} w_{m_2} \phi(z_{m_1})^T \phi(z_{m_2}) + \sum_{n_1=1}^N \sum_{n_2=1}^N \frac{1}{N} \frac{1}{N} \phi(x_{n_1})^T \phi(x_{n_2}) \\
&\quad - 2 \sum_{m=1}^M \sum_{n=1}^N w_m \frac{1}{N} \phi(z_m)^T \phi(x_n) \Bigg)^{1/2} - \Bigg(\sum_{m_1=1}^M \sum_{m_2=1}^M w_{m_1} w_{m_2} k(z_{m_1}, z_{m_2}) \\
&\quad + \sum_{n_1=1}^N \sum_{n_2=1}^N \frac{1}{N} \frac{1}{N} k(x_{n_1}, x_{n_2}) - 2 \sum_{m=1}^M \sum_{n=1}^N w_m \frac{1}{N} k(z_m, x_n) \Bigg)^{1/2} \Bigg|.
\end{aligned} \tag{C.22}$$

Since $\sum_{m=1}^M |w_m| \leq 1$ by construction and $\sum_{n=1}^N \frac{1}{N} = 1$, thanks to the assumption on ϕ this expression is of the form

$$\left| (a + b + 2c)^{1/2} - (A + B + 2C)^{1/2} \right|, \tag{C.23}$$

for suitable $a, A, b, B, c, C \in \mathbb{R}$ with $|a - A|, |b - B|, |c - C| < \delta$. By monotonicity of the square root function, this expression is maximised when $A = a + \delta$, $B = b + \delta$, $C = c + \delta$. Writing $s := a + b + 2C$, we have

$$\begin{aligned}
\left| (a + b + 2c)^{1/2} - (A + B + 2C)^{1/2} \right| &\leq |s^{1/2} - (s + 4\delta)^{1/2}| = (s + 4\delta)^{1/2} - s^{1/2} \\
&\leq s^{1/2} + 2\delta^{1/2} - s^{1/2} = 2\delta^{1/2}.
\end{aligned}$$

□

Theorem 11 Suppose that the random features ϕ converge $\phi(\cdot)^T \phi(\cdot) \to k(\cdot, \cdot)$ uniformly in $\mathcal{X}$ as the number of random features $J \to \infty$. Assume also availability of an approximate Reduced set construction method that solves the minimisation (4.7) either up to a constant multiplicative error, or with an absolute error that can be made arbitrarily small. Then Algorithm 3 outputs a consistent estimator of the kernel mean

embedding μ_X in the sense that

$$\sum_{m=1}^{M} w_m k(z_m, \cdot) \xrightarrow{\mathbb{P}} \mu_X \qquad \text{as } N \to \infty. \tag{C.24}$$

Proof. The output of Algorithm 3 specifies an element $\hat{\mu}_X^{\text{out}} := \sum_{m=1}^{M} w_m k(z_m, \cdot) \in \mathcal{H}$ in the RKHS $\mathcal{H}$ of k. Its RKHS distance to the true kernel mean embedding μ_X of X can be upper bounded by a decomposition using the Triangle inequality, where we write $\hat{\mu}_X^{\phi,\text{out}} := \sum_{m=1}^{M} w_m \phi(z_m)$ for the element of $\mathcal{H}_\phi$ that the Reduced set method constructs to approximate the privacy-protected $\tilde{\mu}_X^\phi$:

$$\begin{aligned}
\left\|\mu_X - \hat{\mu}_X^{\text{out}}\right\|_{\mathcal{H}} &\leq \underbrace{\|\mu_X - \hat{\mu}_X\|_{\mathcal{H}}}_{\text{finite sample error}} + \underbrace{\left\|\hat{\mu}_X - \hat{\mu}_X^{\text{out}}\right\|_{\mathcal{H}}}_{\text{other errors}} \\
&\leq \underbrace{\|\mu_X - \hat{\mu}_X\|_{\mathcal{H}}}_{\text{finite sample error}} + \underbrace{\left|\left\|\hat{\mu}_X^{\phi,\text{out}} - \hat{\mu}_X^{\phi}\right\|_{\mathcal{H}_\phi} - \left\|\hat{\mu}_N - \hat{\mu}_X^{\text{out}}\right\|_{\mathcal{H}}\right|}_{\text{random features error}} + \underbrace{\left\|\hat{\mu}_X^{\phi,\text{out}} - \hat{\mu}_X^{\phi}\right\|_{\mathcal{H}_\phi}}_{\text{other errors}} \\
&\leq \underbrace{\|\mu_X - \hat{\mu}_X\|_{\mathcal{H}}}_{\text{finite sample error}} + \underbrace{\left|\left\|\hat{\mu}_X^{\phi,\text{out}} - \hat{\mu}_X^{\phi}\right\|_{\mathcal{H}_\phi} - \left\|\hat{\mu}_N - \hat{\mu}_X^{\text{out}}\right\|_{\mathcal{H}}\right|}_{\text{random features error}} \\
&\quad + \underbrace{\left\|\hat{\mu}_X^{\phi,\text{out}} - \tilde{\mu}_X^{\phi}\right\|_{\mathcal{H}_\phi}}_{\text{reduced set error}} + \underbrace{\left\|\tilde{\mu}_X^{\phi} - \hat{\mu}_X^{\phi}\right\|_{\mathcal{H}_\phi}}_{\text{privacy error}}.
\end{aligned} \tag{C.25}$$

The finite sample error tends to 0 as $N \to \infty$ in probability by consistency of the empirical kernel mean estimate. The random features error goes to 0 as $N \to \infty$ by Lemma 7, since $J \to \infty$ as $N \to \infty$ and $\phi(\cdot)^T \phi(\cdot) \to k(\cdot, \cdot)$ uniformly in $\mathcal{X}$ as $J \to \infty$. The privacy error goes to 0 as $N \to \infty$ by the same argument as in the proof of Theorem 10, with F replaced by J. So it remains to show that the reduced set error also goes to 0 as $N \to \infty$, in probability.

First, note that the private empirical kernel mean embedding $\hat{\mu}_X^\phi = \frac{1}{N} \sum_{n=1}^{N} \phi(x_n)$ is in the feasible set of the constrained minimisation problem solved by the reduced set method, as the sum of absolute values of weights in this expansion is $N|\frac{1}{N}| = 1 \leq 1$. The RKHS $\mathcal{H}_\phi$ distance of $\hat{\mu}_X^\phi$ to the optimisation target $\tilde{\mu}_X^\phi$ equals the privacy error, so it follows that the reduced set error is upper bounded by the privacy error, and hence also goes to 0 as $N \to \infty$:

$$\underbrace{\left\|\hat{\mu}_X^{\phi,\text{out}} - \tilde{\mu}_X^{\phi}\right\|_{\mathcal{H}_\phi}}_{\text{reduced set error}} \leq \underbrace{\left\|\tilde{\mu}_X^{\phi} - \hat{\mu}_X^{\phi}\right\|_{\mathcal{H}_\phi}}_{\text{privacy error}} \xrightarrow{\mathbb{P}} 0 \text{ as } N \to \infty, \tag{C.26}$$

as required to complete the proof. □

Corollary 14. *Let f be any continuous function. Then whenever k is a c_0-universal kernel, applying f to the points output by Algorithm 3 yields a consistent estimator of the kernel mean embedding $\mu_{f(X)}$ of $f(X)$.*

Proof. Noting that the sum of absolute values of weights w_m output by Algorithm 3 is at most C by construction, in light of Theorem 11 we see that Theorem 1 of Simon-Gabriel et al. (2016) applies and gives the desired conclusion. □

C.1.5 Random Features RKHS Algorithm: Convergence Rate

Proposition 11 Suppose that ϕ is a random feature scheme for the kernel k that converges uniformly on any compact set at a rate $\mathcal{O}_p(J^{-1/2})$ with the number J of random features. Then $J(N)$ can be chosen such that if the employed Reduced set method finds a global optimum of (4.7), Algorithm 3 outputs an element that converges to the true kernel mean embedding μ_X at a rate $\mathcal{O}_p(N^{-1/3})$.

Proof. Equation (C.25) shows that the error $\|\mu_X - \hat{\mu}_X^{\text{out}}\|_{\mathcal{H}}$ between the released element $\hat{\mu}_X^{\text{out}}$ and the true kernel mean embedding μ_X can be upper bounded by the sum of four terms: the finite sample error, the random features error, the reduced set error, and the privacy error. Arguing as in the proof of Proposition 11, the finite sample error vanishes at a rate $\mathcal{O}_p(N^{-1/2})$. The proof of Theorem 11 shows that the reduced set error is upper bounded by the privacy error, which itself vanishes at a rate of $\mathcal{O}_p(\sqrt{J}/N)$ by the argument given in the proof of Theorem 10, with F replaced by J. Lemma 7 implies that if the random features converge uniformly at a rate $\mathcal{O}_p(J^{-1/2})$, then the random features error vanishes at a rate $\mathcal{O}_p(J^{-1/4})$. The total convergence rate is thus

$$\mathcal{O}_p\left(N^{-1/2} + \frac{\sqrt{J}}{N} + J^{-1/4}\right), \tag{C.27}$$

and we can check that this becomes $\mathcal{O}_p(N^{-1/3})$ by setting $J = \lfloor N^{4/3} \rfloor$. □

C.1.6 Random Features RKHS Algorithm: Privacy

Proposition 14 Assume that the random feature vectors produced by ϕ are bounded by 1 in L_2 norm ($\|\phi(x)\|_2 \leq 1$ for all $x \in \mathcal{X}$). Then Algorithm 3 is (ε, δ)-differentially private.

Proof. The output of the algorithm is produced by a Reduced set method that is initialised blindly to the database and optimises RKHS distance to the element $\tilde{\mu}_X^\phi \in \mathcal{H}_\phi$, while only having access to the distance to it, rather than any representation of $\tilde{\mu}_X^\phi$. As $\tilde{\mu}_X^\phi$ can be seen as a vector in $\mathbb{R}^J$ obtained by perturbing $\hat{\mu}_X^\phi$ using the Gaussian mechanism with $\Delta_2 = \frac{2}{N}$, it suffices to show that the L_2-sensitivity of $\hat{\mu}_X^\phi$ is upper bounded by $\frac{2}{N}$. To this end, assume $D = \{x_1, \ldots, x_N\}$ and $D' = \{x'_1, \ldots, x'_N\}$ are two neighbouring databases of cardinality N, differing w.l.o.g. in their last element only. Then

$$\begin{aligned} \|\hat{\mu}_D^\phi - \hat{\mu}_{D'}^\phi\|_2 &= \left\| \frac{1}{N} \sum_{n=1}^{N} \phi(x_n) - \frac{1}{N} \sum_{n=1}^{N} \phi(x'_n) \right\|_2 \\ &= \frac{1}{N} \|\phi(x_N) - \phi(x'_N)\|_2 \\ &\leq \frac{1}{N} \|\phi(x_N)\|_2 + \frac{1}{N} \|\phi(x'_N)\|_2 \leq \frac{2}{N}, \end{aligned} \tag{C.28}$$

as required to complete the proof. □

C.2 Setup of Empirical Illustrations

We considered two scenarios in our basic empirical evaluations shown in Sections 4.4 and 4.5:

1. *No publishable subset*: No rows of the private database are, or can be made public without some privacy-ensuring modification.

2. *Publishable subset*: A small part of the private database is already public, or can be made public, perhaps for one of the several possible reasons outlined in Section 4.1.

To illustrate the impact of data dimensionality on the performance of the proposed algorithms, we provide results on datasets with data dimension $D = 2$ and $D = 5$. In both cases we constructed a synthetic private dataset by sampling $N = 100,000$ data points from a multivariate Gaussian mixture distribution. The mixture had 10 components, with mixing weights proportional to $1, \frac{1}{2}, \ldots, \frac{1}{10}$, and the means of the components were chosen randomly themselves from a spherical Gaussian distribution with mean $[100, \ldots, 100]$ and covariance $200 I_D$. Each of the N private data points was simulated by first sampling its mixture component using the mixing weights as

probabilities, and then the point itself was sampled from a spherical Gaussian centered at the mean of the chosen mixture component and with covariance $30I_D$.

We chose to work with the widely popular exponentiated quadratic kernel $k(x_1, x_2) = e^{-\gamma\|x_1 - x_2\|_2^2}$ for $\mathbb{R}^D$-valued data (also known as a Gaussian kernel, or a squared exponential kernel), with the parameter setting $\gamma = 10^{-4}/D$. This kernel is known to be *characteristic* Fukumizu et al. (2008), and so as discussed in Section 4.2.2, no information about the data generating distribution p_X is lost by working with its kernel mean embedding μ_X.

We used our proposed algorithms to release an approximate version of the empirical KME of the private database, in such a way that the output satisfies the definition of (ε, δ)-differential privacy. We investigated the common privacy levels given by $\varepsilon \in \{0.01, 0.1, 1.0\}$, and used the fixed value of $\delta = 10^{-6}$, which satisfies the usual requirement that $\delta \ll \frac{1}{N}$.

C.2.1 Evaluation Metric

The geometry of the RKHS $\mathcal{H}$ allows comparing the performance of different algorithms by computing the RKHS distance Δ between the empirical KME $\hat{\mu}_X$ computed using all N private data points (and which could not have been released without violating differential privacy) and the element of the RKHS represented by the actually released weighted set of synthetic data points $(z_1, w_1), \ldots, (z_M, w_M)$:

$$\Delta := \left\| \hat{\mu}_X - \sum_{m=1}^{M} w_m k(z_m, \cdot) \right\|_{\mathcal{H}}.$$

Moreover, as the empirical KME $\hat{\mu}_X$ is based on a large sample size of $N = 100,000$ i.i.d. data points, it can be expected to be a good proxy for the true KME μ_X of the data-generating random variable X. In that case Δ is also a good proxy for the RKHS distance between the true KME μ_X and the RKHS element represented by the released dataset.

C.2.2 Scenario 1: No Publishable Subset

Algorithm 2 requires specifying the synthetic data points $z_1, \ldots, z_M$ in advance, before seeing the private data. If no part of the private data has already been published (which could then be used for the synthetic data points), one can construct the synthetic data points by sampling them randomly from a suitable probability distribution q.

For the consistency result of Theorem 10 to apply, the support of q must include all possible private data points. In our case the private data takes values in $\mathbb{R}^D$, and so this requirement is satisfied by any distribution on $\mathbb{R}^D$ with full support. We used a spherical Gaussian distribution $q = \mathcal{N}(0, \sigma_q I_D)$ with $\sigma_q = 500$ for sampling the synthetic data points.

The implementation of Algorithm 3 used $J = 10,000$ random features for accurate approximation of the kernel, and an iterative gradient-based optimisation procedure to solve the reduced set problem (Equation (4.7) in Algorithm 3).

Figure 4.2 shows how the RKHS distance Δ changes with the number of synthetic data points M, for different requested privacy level ε for Algorithm 2 (solid lines) and Algorithm 3 (dashed lines), on datasets with dimensionality $D = 2$ (left subfigure) and $D = 5$ (right subfigure). We observe that the additional ability of Algorithm 3 to optimise the *locations* of the synthetic data points (rather than just the weights, as is the case for Algorithm 2) is more helpful in the higher-dimensional case $D = 5$, where the randomly sampled synthetic data points are less likely to land close to private data points.

C.2.3 Scenario 2: Publishable Subset

Here we explored the interesting scenario where one can exploit the fact that a small part of the private database is actually public, and use the public rows as the fixed synthetic data points in Algorithm 2. Specifically, we assume (without loss of generality) that the first M rows of the private database (where $M \ll N$) are public, and we take the synthetic data points to be $z_1 \leftarrow x_1, \ldots, z_M \leftarrow x_M$.

Observe that in this case $\hat{\mu}^{\text{baseline}} := \frac{1}{M}\sum_{m=1}^{M} k(z_m, \cdot)$, i.e., uniform weighting of the synthetic data points, is already expected to be a strong baseline since $\hat{\mu}^{\text{baseline}}$ is itself a consistent estimator of μ_X, (although based on a much smaller sample size $M \ll N$). The purpose of Algorithm 2 is to find (generally non-uniform) $w_1, \ldots, w_M$ that reweight the public data points using the information in the large private dataset, but respecting differential privacy. Figure 4.1 shows how the RKHS distance Δ changes with the number of public data points M, for different privacy levels ε.

For comparison, the figures also show the RKHS distances Δ achieved by the baseline that simply weights the public points uniformly. We can see that in both cases $D = 2$ and $D = 5$, if the ratio of public to private points is low enough, Algorithm 2 provides a substantial benefit over this baseline (note the logarithmic scaling). Since usually obtaining permission to publish a larger subset of the private data unchanged

will come at an increased cost, the ability to instead reweight a smaller public dataset using Algorithm 2 in a differentially private manner is useful.

Appendix D

Program Synthesis

This Appendix contains material supplementing the content of Chapter 5, which presented *DeepCoder*, an instantiation of a framework to accelerate a difficult discrete *search* problem arising in program synthesis through gradient-based optimization of neural networks on a supervised learning task.

D.1 Example Programs

This section shows example programs in our Domain Specific Language (DSL) defined in Section 5.4.1, together with input-output examples and short illustrative descriptions (that are not used by DeepCoder). These programs have been inspired by simple tasks appearing on real programming competition websites, and are meant to illustrate the expressive power of our DSL.

Program 1:	**IO example:**	*Description*:
`k` ← `int`	*Input*:	A new shop near you is selling n paintings. You have $k < n$ friends and you would like to buy each of your friends a painting from the shop. Return the minimal amount of money you will need to spend.
`b` ← `[int]`	`2, [3 5 4 7 5]`	
`c` ← SORT `b`	*Output*:	
`d` ← TAKE `k c`	`[7]`	
`e` ← SUM `d`		

Program 2:

```
w ← [int]
t ← [int]
c ← Map (*3) w
d ← ZipWith (+) c t
e ← Maximum d
```

IO example:

Input:
`[6 2 4 7 9]`, `[5 3 6 1 0]`
Output:
`27`

Description:
In soccer leagues, match winners are awarded 3 points, losers 0 points, and both teams get 1 point in the case of a tie. Compute the number of points awarded to the winner of a league given two arrays w, t of the same length, where $w[i]$ (resp. $t[i]$) is the number of times team i won (resp. tied).

Program 3:

```
a ← [int]
b ← [int]
c ← ZipWith (-) b a
d ← Count (>0) c
```

IO example:

Input:
`[6 2 4 7 9]`, `[5 3 2 1 0]`
Output:
`4`

Description:
Alice and Bob are comparing their results in a recent exam. Given their marks per question as two arrays a and b, count on how many questions Alice got more points than Bob.

Program 4:

```
h ← [int]
b ← Scanl1 Min h
c ← ZipWith (-) h b
d ← Filter (>0) c
e ← Sum d
```

IO example:

Input:
`[8 5 7 2 5]`
Output:
`5`

Description:
Perditia is very peculiar about her garden and wants that the trees standing in a row are all of non-increasing heights. Given the tree heights in centimeters in order of the row as an array `h`, compute how many centimeters she needs to trim the trees in total.

Program 5:

```
x ← [int]
y ← [int]
c ← Sort x
d ← Sort y
e ← Reverse d
f ← ZipWith (*) d e
g ← Sum f
```

IO example:

Input:
`[7 3 8 2 5]`, `[2 8 9 1 3]`
Output:
`79`

Description:
Xavier and Yasmine are laying sticks to form non-overlapping rectangles on the ground. They both have fixed sets of pairs of sticks of certain lengths (represented as arrays `x` and `y` of numbers). Xavier only lays sticks parallel to the x axis, and Yasmine lays sticks only parallel to y axis. Compute the area their rectangles will cover at least.

Program 6:

```
a ← [int]
b ← Reverse a
c ← ZipWith Min a b
```

IO example:

Input:

```
[3 7 5 2 8]
```

Output:

```
[3 2 5 2 3]
```

Description:

A sequence called Billy is looking into the mirror, wondering how much weight it could lose by replacing any of its elements by their mirror images. Given a description of Billy as an array b of length n, return an array c of minimal sum where each element $c[i]$ is either $b[i]$ or its mirror image $b[n-i-1]$.

Program 7:

```
t ← [int]
p ← [int]
c ← Map (-1) t
d ← Map (-1) p
e ← ZipWith (+) c d
f ← Minimum e
```

IO example:

Input:

```
[4 8 11 2],
[2 3 4 1]
```

Output:

```
1
```

Description:

Umberto has a large collection of ties and matching pocket squares—too large, his wife says—and he needs to sell one pair. Given their values as arrays `t` and `p`, assuming that he sells the cheapest pair, and selling costs `2`, how much will he lose from the sale?

Program 8:

```
s ← [int]
p ← [int]
c ← Scanl1 (+) p
d ← ZipWith (*) s c
e ← Sum d
```

IO example:

Input:

```
[4 7 2 3],
[2 1 3 1]
```

Output:

```
62
```

Description:

Zack always promised his n friends to buy them candy, but never did. Now he won the lottery and counts how often and how much candy he promised to his friends, obtaining arrays `p` (number of promises) and `s` (number of promised sweets). He announces that to repay them, he will buy `s[1]+s[2]+...+s[n]` pieces of candy for the first `p[1]` days, then `s[2]+s[3]+...+s[n]` for `p[2]` days, and so on, until he has fulfilled all promises. How much candy will he buy in total?

Program 9:

```
s ← [int]
b ← REVERSE s
c ← ZIPWITH (-) b s
d ← FILTER (>0) c
e ← SUM d
```

IO example:

Input:

```
[1 2 4 5 7]
```

Output:

```
9
```

Description:

Vivian loves rearranging things. Most of all, when she sees a row of heaps, she wants to make sure that each heap has more items than the one to its left. She is also obsessed with efficiency, so always moves the least possible number of items. Her dad really dislikes if she changes the size of heaps, so she only moves single items between them, making sure that the set of sizes of the heaps is the same as at the start; they are only in a different order. When you come in, you see heaps of sizes (of course, sizes strictly monotonically increasing) `s[0], s[1], ... s[n]`. What is the maximal number of items that Vivian could have moved?

Figure D.1 shows the predictions made by a neural network trained on programs of length $T = 4$ that were ensured to be semantically disjoint from all 9 example programs shown in this section. For each task, the neural network was provided with 5 input-output examples.

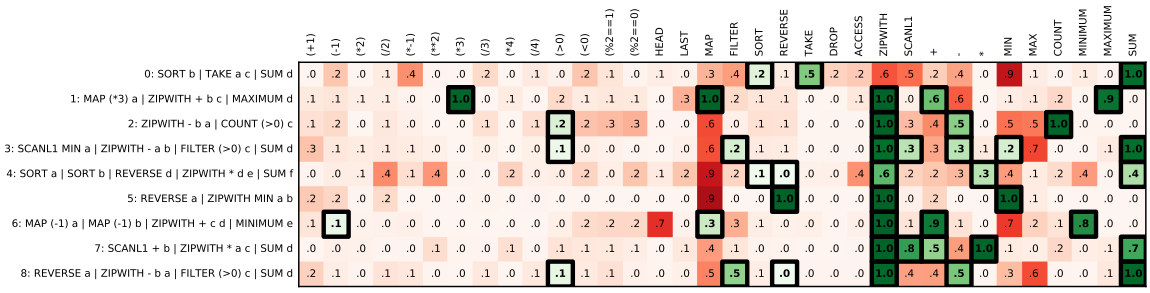


Fig. D.1 Predictions of a neural network on the 9 example programs described in this section. Numbers in squares would ideally be close to 1 (function is present in the ground truth source code), whereas all other numbers should ideally be close to 0 (function is not needed).

D.2 Experimental Results

Results presented in Section 5.5.1 showcased the computational speedups obtained from the LIPS framework (using DeepCoder), as opposed to solving each program synthesis problem with only the information about global incidence of functions in source code available. For completeness, here we show plots of raw computation times of each search procedure to solve a given number of problems.

Figure D.2 shows the computation times of DFS, of Enumerative search with a *Sort and add* scheme, of the λ^2 and Sketch solvers with a *Sort and add* scheme, and of Beam search, when searching for a program consistent with input-output examples generated from $P = 500$ different test programs of length $T = 3$. As discussed in Section 5.5.1, these test programs were ensured to be semantically disjoint from all programs used to train the neural networks, as well as from all programs of shorter length (as discussed in Section 5.4.2).

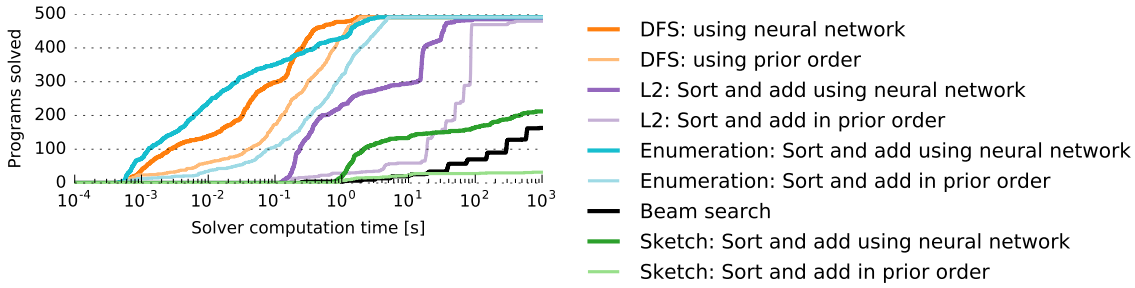


Fig. D.2 Number of test problems solved versus computation time.

The "steps" in the results for Beam search are due to our search strategy, which doubles the size of the considered beam until reaching the timeout (of 1000 seconds) and thus steps occur whenever the search for a beam of size 2^k is finished. For λ^2, we observed that no solution for a given set of allowed functions was ever found after about 5 seconds (on the benchmark machines), but that λ^2 continued to search. Hence, we introduced a hard timeout after 6 seconds for all but the last iterations of our *Sort and add* scheme.

Figure D.3 shows the computation times of DFS, Enumerative search with a *Sort and add* scheme, and λ^2 with a *Sort and add* scheme when searching for programs consistent with input-output examples generated from $P = 100$ different test programs of length $T = 5$. The neural network was trained on programs of length $T = 4$.

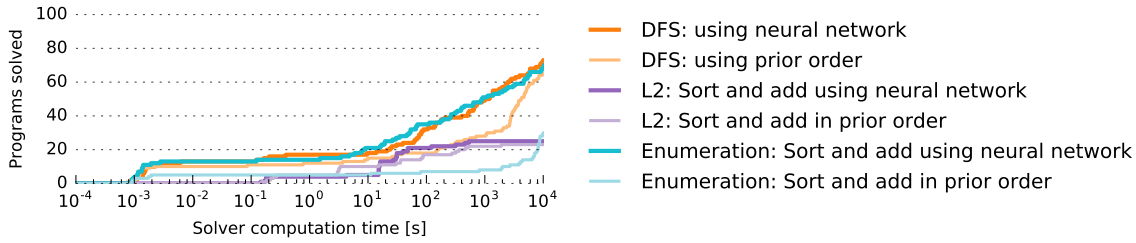


Fig. D.3 Number of test problems solved versus computation time.

D.3 The Neural Network

As briefly described in Section 5.4.3, we used the following simple feed-forward architecture encoder:

- For each input-output example in the set generated from a single ground truth program:
  - Pad arrays appearing in the inputs and in the output to a maximum length $L = 20$ with a special NULL value.
  - Represent the type (singleton integer or integer array) of each input and of the output using a one-hot-encoding vector. Embed each integer in the valid integer range (-256 to 255) using a learned embedding into $E = 20$ dimensional space. Also learn an embedding for the padding NULL value.
  - Concatenate the representations of the input types, the embeddings of integers in the inputs, the representation of the output type, and the embeddings of integers in the output into a single (fixed-length) vector.
  - Pass this vector through $H = 3$ hidden layers containing $K = 256$ sigmoid units each.
- Pool the last hidden layer encodings of each input-output example together by simple arithmetic averaging.

Figure D.4 shows a schematic drawing of this encoder architecture, together with the decoder that performs independent binary classification for each function in the DSL, indicating whether or not it appears in the ground truth source code.

While DeepCoder learns to embed integers into a $E = 20$ dimensional space, we built the system up gradually, starting with a $E = 2$ dimensional space and only training on programs of length $T = 1$. Such a small scale setting allowed easier

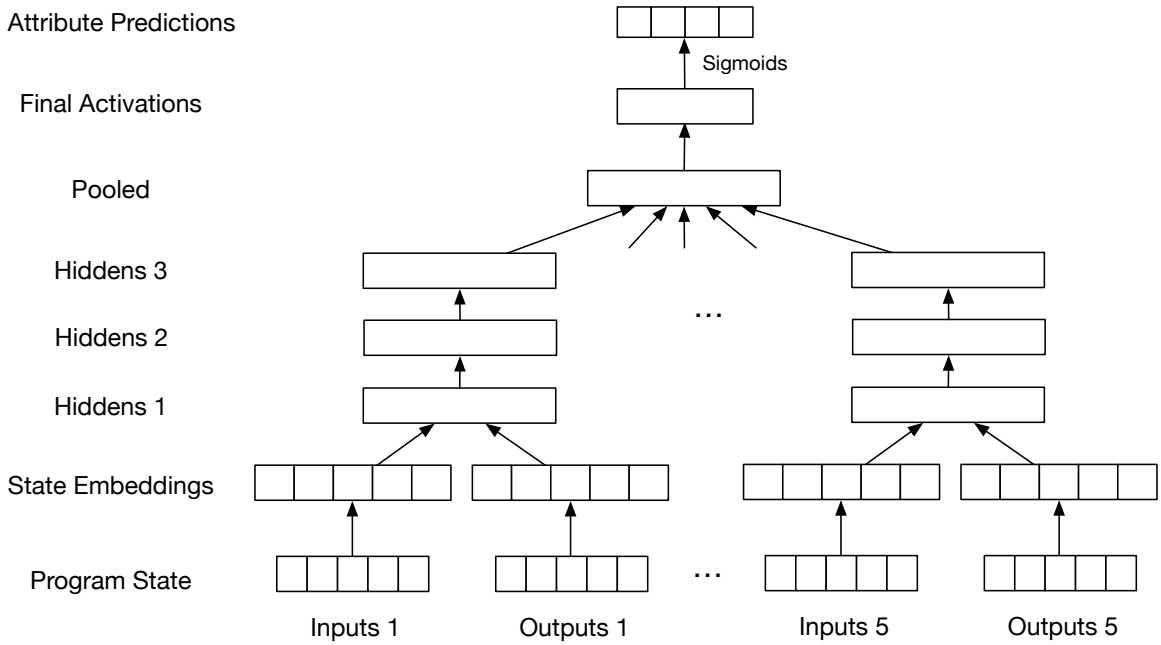


Fig. D.4 Schematic representation of our feed-forward encoder, and the decoder.

investigation of the workings of the neural network, and indeed Figure D.5 below shows a learned embedding of integers in $\mathbb{R}^2$. The figure demonstrates that the network has learnt the concepts of number magnitude, sign (positive or negative) and evenness, presumably due to FILTER `(>0)`, FILTER `(<0)`, FILTER `(%2==0)` and FILTER `(%2==1)` all being among the programs on which the network was trained.

D.4 Depth-First Search

We use an optimized C++ implementation of depth-first search (DFS) to search over programs with a given maximum length T. In depth-first search, we start by choosing the first function (and its arguments) of a potential solution program, and then recursively consider all ways of filling in the rest of the program (up to length T), before moving on to a next choice of first instruction (if a solution has not yet been found).

A program is considered a solution if it is consistent with all $M = 5$ provided input-output examples. Note that this requires evaluating all candidate programs on the M inputs and checking the results for equality with the provided M respective outputs. Our implementation of DFS exploits the sequential structure of programs in our DSL by caching the results of evaluating all prefixes of the currently considered

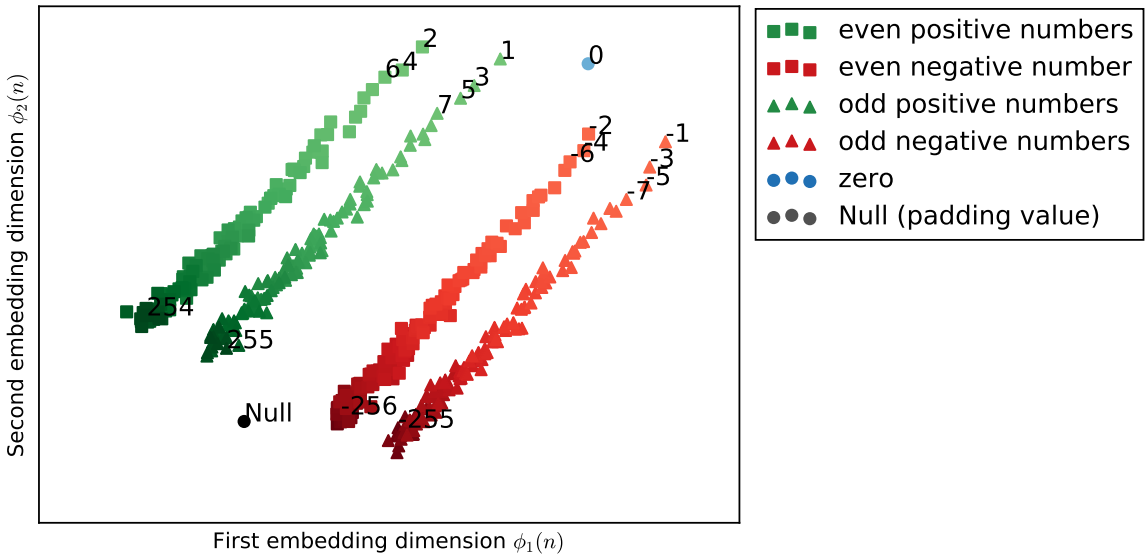


Fig. D.5 A learned embedding of integers $\{-256, -255, \ldots, -1, 0, 1, \ldots, 255\}$ in $\mathbb{R}^2$. The color intensity corresponds to the magnitude of the embedded integer.

program on the example inputs, thus allowing efficient reuse of computation between candidate programs with common prefixes.

This allows us to explore the search space at roughly the speed of $\sim 3 \times 10^6$ programs per second.

When the search procedure extends a partial program by a new function, it has to try the functions in the DSL in some order. At this point DFS can opt to consider the functions as ordered by their predicted probabilities from the neural network. The probability of a function consisting of a higher-order function and a lambda is taken to be the minimum of the probabilities of the two constituent functions.

D.5 Training Loss Function

In Section 5.4.5 we outlined a justification for using marginal probabilities of individual functions as a sensible intermediate representation to provide a solver employing a *Sort and add* scheme (we considered Enumerative search and the Sketch solver with this scheme). Here we provide a more detailed discussion.

Predicting program components from input-output examples can be cast as a multilabel classification problem, where each instance (set of input-output examples) is associated with a set of relevant labels (functions appearing in the code that generated

the examples). We denote the number of labels (functions) by C, and note that throughout this work $C = 34$.

When the task is to predict a subset of labels $\mathbf{y} \in \{0,1\}^C$, different loss functions can be employed to measure the prediction error of a classifier $\mathbf{h}(\mathbf{x})$ or ranking function $\mathbf{f}(\mathbf{x})$. Dembczynski et al. (2010) discuss the following three loss functions:

- *Hamming loss* counts the number of labels that are predicted incorrectly by a classifier $\mathbf{h}$:
$$L_H(\mathbf{y}, \mathbf{h}(\mathbf{x})) = \sum_{c=1}^{C} \mathbb{1}_{\{y_c \neq h_c(\mathbf{x})\}}. \tag{D.1}$$

- *Rank loss* counts the number of label pairs violating the condition that relevant labels are ranked higher than irrelevant ones by a scoring function $\mathbf{f}$:
$$L_r(\mathbf{y}, \mathbf{f}(\mathbf{x})) = \sum_{(i,j):y_i=1,y_j=0}^{C} \mathbb{1}_{\{f_i<f_j\}}. \tag{D.2}$$

- *Subset Zero-One loss* indicates whether all labels have been correctly predicted by $\mathbf{h}$:
$$L_s(\mathbf{y}, \mathbf{h}(\mathbf{x})) = \mathbb{1}_{\{\mathbf{y}\neq\mathbf{h}(\mathbf{x})\}}. \tag{D.3}$$

Dembczynski et al. (2010) proved that Bayes optimal decisions under the Hamming and Rank loss functions, i.e., decisions minimizing the expected loss under these loss functions, can be computed from marginal probabilities $p_c(y_c|\mathbf{x})$. This *suggests* that:

- Multilabel classification under these two loss functions may not benefit from considering dependencies between the labels.

- "Instead of minimizing the Rank loss directly, one can simply use any approach for single label prediction that properly estimates the marginal probabilities." (Dembczyński et al., 2012)

Training the neural network with the negative cross entropy loss function as the training objective is precisely a method for properly estimating the marginal probabilities of labels (functions appearing in source code). It is thus a sensible step in preparation for making predictions under a Rank loss.

It remains to discuss the relationship between the Rank loss and the actual quantity we care about, which is the total runtime of a *Sort and add* search procedure. Recall the simplifying assumption that the runtime of searching for a program of length T

with C functions made available to the search is proportional to C^T, and consider a *Sort and add* search for a program of length T, where the size of the active set is increased by 1 whenever the search fails. Starting with an active set of size 1, the total time until a solution is found can be upper bounded by

$$1^T + 2^T + \cdots + C_A^T \leq C_A^{T+1} \leq CC_A^T,$$

where C_A is the size of the active set when the search finally succeeds (i.e., when the active set finally contains all necessary functions for a solution to exist). Hence the total runtime of a *Sort and add* search can be upper bounded by a quantity that is proportional to C_A^T.

Now fix a valid program solution P that requires C_P functions, and let $\mathbf{y}_P \in \{0,1\}^C$ be the indicator vector of functions used by P. Let $D := C_A - C_P$ be the number of redundant operations added into the active set until all operations from P have been added.

Example 15. Suppose the labels, as sorted by decreasing predicted marginal probabilities $\mathbf{f}(\mathbf{x})$, are as follows:

1 1 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Then the solution P contains $C_P = 6$ functions, but the active set needs to grow to size $C_A = 11$ to include all of them, adding $D = 5$ redundant functions along the way. Note that the rank loss of the predictions $\mathbf{f}(\mathbf{x})$ is $L_r(\mathbf{y}_P, \mathbf{f}(\mathbf{x})) = 2 + 5 = 7$, as it double counts the two redundant functions which are scored higher than two relevant labels.

Noting that in general $L_r(\mathbf{y}_P, \mathbf{f}(\mathbf{x})) \geq D$, the previous upper bound on the runtime of *Sort and add* can be further upper bounded as follows:

$$C_A^T = (C_P + D)^T \leq \text{const} + \text{const} \times D^T \leq \text{const} + \text{const} \times L_r(\mathbf{y}_P, \mathbf{f}(\mathbf{x}))^T.$$

Hence we see that for a constant value of T, this upper bound can be minimized by optimizing the Rank loss of the predictions $\mathbf{f}(\mathbf{x})$. Note also that $L_r(\mathbf{y}_P, \mathbf{f}(\mathbf{x})) = 0$ would imply $D = 0$, in which case $C_A = C_P$.

D.6 Domain Specific Language of DeepCoder

Here we provide a description of the semantics of our DSL from Section 5.4.1, both in English and as a Python implementation. Throughout, NULL is a special value that can be set e.g. to an integer outside the working integer range.

First-order functions:

- HEAD :: `[int] -> int`
  `lambda xs: xs[0] if len(xs)>0 else Null`
  Given an array, returns its first element (or NULL if the array is empty).

- LAST :: `[int] -> int`
  `lambda xs: xs[-1] if len(xs)>0 else Null`
  Given an array, returns its last element (or NULL if the array is empty).

- TAKE :: `int -> [int] -> int`
  `lambda n, xs: xs[:n]`
  Given an integer `n` and array `xs`, returns the array truncated after the `n`-th element. (If the length of `xs` was no larger than `n` in the first place, it is returned without modification.)

- DROP :: `int -> [int] -> int`
  `lambda n, xs: xs[n:]`
  Given an integer `n` and array `xs`, returns the array with the first `n` elements dropped. (If the length of `xs` was no larger than `n` in the first place, an empty array is returned.)

- ACCESS :: `int -> [int] -> int`
  `lambda n, xs: xs[n] if n>=0 and len(xs)>n else Null`
  Given an integer `n` and array `xs`, returns the (`n`+1)-st element of `xs`. (If the length of `xs` was less than or equal to `n`, the value NULL is returned instead.)

- MINIMUM :: `[int] -> int`
  `lambda xs: min(xs) if len(xs)>0 else Null`
  Given an array, returns its minimum (or NULL if the array is empty).

- MAXIMUM :: `[int] -> int`
  `lambda xs: max(xs) if len(xs)>0 else Null`
  Given an array, returns its maximum (or NULL if the array is empty).

- REVERSE `:: [int] -> [int]`
  `lambda xs: list(reversed(xs))`
  Given an array, returns its elements in reversed order.

- SORT `:: [int] -> [int]`
  `lambda xs: sorted(xs)`
  Given an array, return its elements in non-decreasing order.

- SUM `:: [int] -> int`
  `lambda xs: sum(xs)`
  Given an array, returns the sum of its elements. (The sum of an empty array is 0.)

Higher-order functions:

- MAP `:: (int -> int) -> [int] -> [int]`
  `lambda f, xs: [f(x) for x in xs]`
  Given a lambda function `f` mapping from integers to integers, and an array `xs`, returns the array resulting from applying `f` to each element of `xs`.

- FILTER `:: (int -> bool) -> [int] -> [int]`
  `lambda f, xs: [x for x in xs if f(x)]`
  Given a predicate `f` mapping from integers to truth values, and an array `xs`, returns the elements of `xs` satisfying the predicate in their original order.

- COUNT `:: (int -> bool) -> [int] -> int`
  `lambda f, xs: len([x for x in xs if f(x)])`
  Given a predicate `f` mapping from integers to truth values, and an array `xs`, returns the number of elements in `xs` satisfying the predicate.

- ZIPWITH `:: (int -> int -> int) -> [int] -> [int] -> [int]`
  `lambda f, xs, ys: [f(x, y) for (x, y) in zip(xs, ys)]`
  Given a lambda function `f` mapping integer pairs to integers, and two arrays `xs` and `ys`, returns the array resulting from applying `f` to corresponding elements of `xs` and `ys`. The length of the returned array is the minimum of the lengths of `xs` and `ys`.

- SCANL1 `:: (int -> int -> int) -> [int] -> [int]`
  Given a lambda function `f` mapping integer pairs to integers, and an array `xs`,

returns an array `ys` of the same length as `xs` and with its content defined by the recurrence `ys[0] = xs[0]`, `ys[n] = f(ys[n-1], xs[n])` for $n \geq 1$.

The Int→Int lambdas `(+1)`, `(-1)`, `(*2)`, `(/2)`, `(*(-1))`, `(**2)`, `(*3)`, `(/3)`, `(*4)`, `(/4)` provided by our DSL map integers to integers in a self-explanatory manner. The Int→Bool lambdas `(>0)`, `(<0)`, `(%2==0)`, `(%2==1)` respectively test positivity, negativity, evenness and oddness of the input integer value. Finally, the Int→Int→Int lambdas `(+)`, `(-)`, `(*)`, Min, Max apply a function to a pair of integers and produce a single integer.

As an example, consider the function Scanl1 Max, consisting of the higher-order function Scanl1 and the Int→Int→Int lambda Max. Given an integer array `a` of length L, this function computes the running maximum of the array `a`. Specifically, it returns an array `b` of the same length L whose i-th element is the maximum of the first i elements in `a`.

D.7 Analysis of Trained Neural Networks

We analyzed the performance of trained neural networks by investigating which program instructions tend to get confused by the networks. To this end, we looked at a generalization of confusion matrices to the multilabel classification setting: for each attribute in a ground truth program (rows) measure how likely each other attribute (columns) is predicted as a false positive. More formally, in this matrix the (i, j)-entry is the average predicted probability of attribute j among test programs that *do* possess attribute i and *do not* possess attribute j. Intuitively, the i-th row of this matrix shows how the presence of attribute i confuses the network into incorrectly predicting each other attribute j.

Figure D.6 shows this conditional confusion matrix for the neural network and $P = 500$ program test set configuration used to obtain Table 5.1. We re-ordered the confusion matrix to try to expose block structure in the false positive probabilities, revealing groups of instructions that tend to be difficult to distinguish. Figure D.7 show the conditional confusion matrix for the neural network used to obtain the table in Figure 5.3a. While the results are somewhat noisy, we observe a few general tendencies:

- There is increased confusion amongst instructions that select out a single element from an array: Head, Last, Access, Minimum, Maximum.

- Some common attributes get predicted more often regardless of the ground truth program: FILTER, `(>0)`, `(<0)`, `(%2==1)`, `(%2==0)`, MIN, MAX, `(+)`, `(-)`, ZIPWITH.

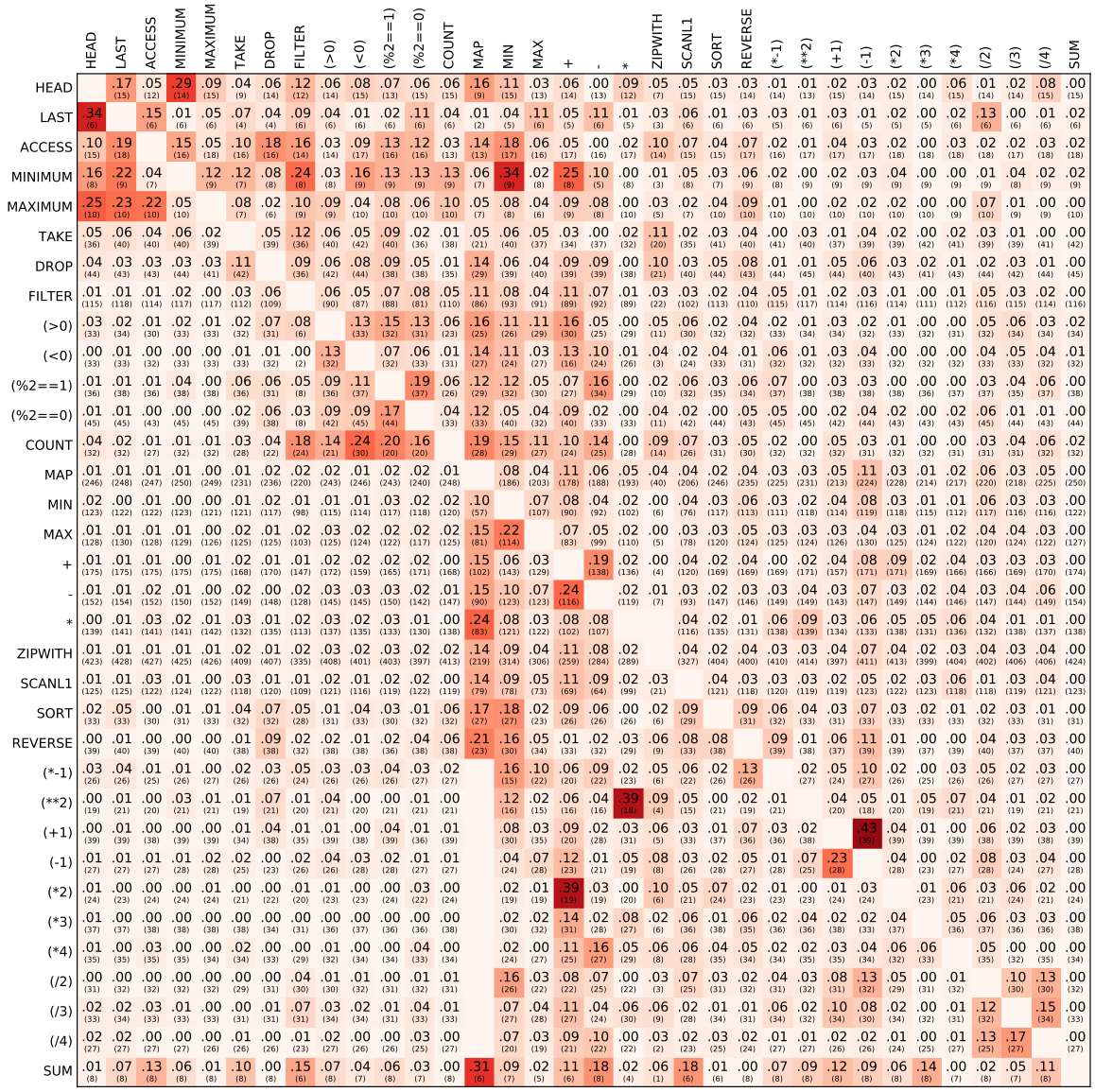


Fig. D.6 Conditional confusion matrix for the neural network and test set of $P = 500$ programs of length $T = 3$ that were used to obtain the results presented in Table 5.1. Each cell contains the average false positive probability (in larger font) and the number of test programs from which this average was computed (smaller font, in brackets). The color intensity of each cell's shading corresponds to the magnitude of the average false positive probability.

- There are some groups of lambdas that are more difficult for the network to distinguish within: `(+)` vs `(-)`; `(+1)` vs `(-1)`; `(/2)` vs `(/3)` vs `(/4)`.
- When a program uses `(**2)`, the network often thinks it's using `(*)`, presumably because both can lead to large values in the output.

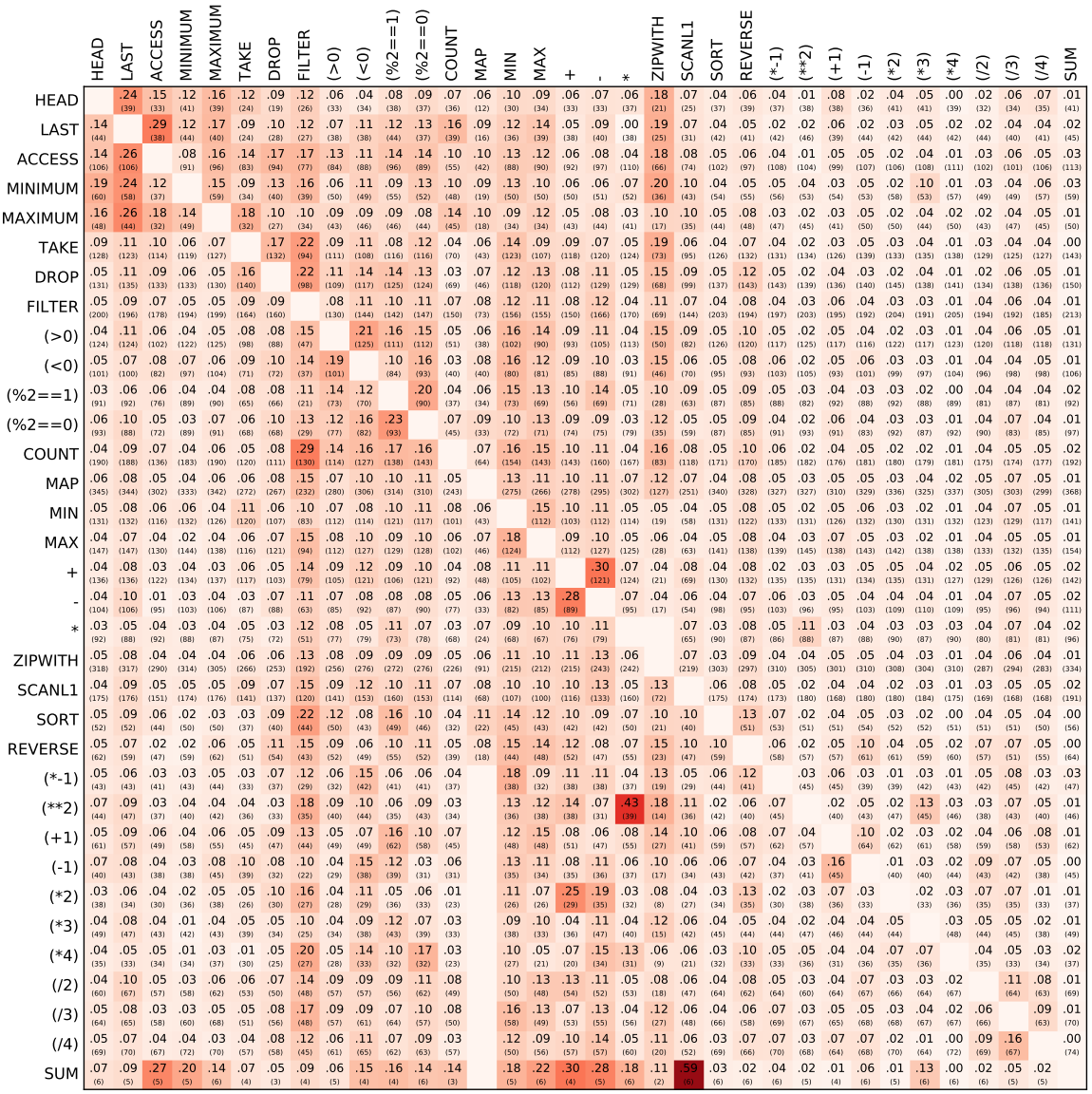


Fig. D.7 Conditional confusion matrix for the neural network and test set of $P = 500$ programs of length $T = 5$. The presentation is the same as in Figure D.6.