Practical Methods for Automated Algorithm Design in Machine Learning and Computational Biology

Quang Minh Hoang

CMU-CS-23-139

October 2023

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Carl Kingsford (Chair)
David P. Woodruff
Maria-Florina Balcan
Risto Miikkulainen (The University of Texas at Austin)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the Gordon and Betty Moore Foundation under award number 4554, UPMC through the Center for Machine Learning and Health, Schmidt Sciences, the National Science Foundation under award number 197540, and the National Institute of Health under awards numbered R01GM122935 and R01HG102470. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Automated Machine Learning, Computational Biology, Combinatorial Optimization, Sequence Sketching, Federated Learning, Meta Learning

For my beloved Minh Tran.
This would not have been possible without your endless encouragement.

Abstract

Configuration tuning is an essential practice to achieve good performance with many computational methods. However, configuring complex and discrete algorithms often requires significant trial-and-error effort due to a lack of automated solutions. In large-scale systems where computational tasks are numerous and constantly changing in specificity, the repetitive cost of manual tuning becomes a major bottleneck that hinders scalability. Moreover, the absence of a systematic approach to configure deployment settings makes it challenging to replicate the obtained results in different deploying conditions. To address these problems, this thesis focuses on developing new data-driven automated algorithm design (AAD) frameworks in several classical and multi-task settings. Specifically, in the classical configuration tuning setting, we address the problems of kernel selection for Bayesian methods, and minimizer construction for biological sequence sketching. In the multi-task scenario, we address the problems of privacy-preserving neural architecture search for multiple clients, and meta-learning for parameter optimization in a heterogeneous task stream. In all of these problems, the variables to be optimized often have underlying discrete structures such as trees, graphs or permutations. Our contribution is a suite of reformulation techniques that result in efficient and accurate tuning methods for these configuration domains. Finally, we demonstrate the performance of our methods on practical scenarios and show that they have significantly outperformed state-of-the-art benchmarks.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Carl Kingsford, for his guidance, invaluable insights, and unwavering support throughout my journey. His tutelage and wisdom have played a pivotal role in shaping not only my research and academic growth but also my perspectives in life. This thesis would not have become a reality without his endless patience and brilliant technical mind.

I am thankful to the members of my thesis committee, Dr. David Woodruff, Dr. Nina Balcan, and Dr. Risto Miikkulainen, for their expert feedback, constructive criticism, and the thoughtful discussions that enriched the quality of my research. I'm also grateful to my fellow researchers and collaborators, both past and present members of the Kingsford group. These are the individuals who have shared their knowledge with me, collaborated on projects, and provided a supportive community that enhanced my research experience at CMU.

I am deeply indebted to my family, especially my parents, for their unwavering support. I am grateful to my partner, Minh Tran, who has stood by my side throughout this long and challenging journey, as well as to my brother, Nghia Hoang, for his companionship and intellectual assistance. Their unwavering belief in my abilities has been a constant source of motivation. I thank them for being there during both my best and worst moments.

Last but not least, I genuinely appreciate my dearest friends who have supported, encouraged, and positively influenced me throughout my life. To my Brothers for Life, thank you for being the closest thing to family when I first ventured out into the world. You have been, and will always be, special in my heart. To the beloved Pittsburgh Penguins, thank you for making my time here so much brighter and filled with joy. Without you, I could not have imagined having the strength to persist through this journey.

Contents

1 Introduction **1**
- 1.1 Classical methods for automated algorithm design . . . 2
  - 1.1.1 Heuristic search . . . 2
  - 1.1.2 Evolutionary strategies . . . 3
  - 1.1.3 Sequential model-based optimization . . . 4
- 1.2 Domain-specific AAD Problems . . . 5
- 1.3 Thesis aims and contributions . . . 8

2 Kernel Selection **11**
- 2.1 Introduction . . . 11
- 2.2 Problem setting . . . 14
- 2.3 Methodology . . . 16
  - 2.3.1 Reformulating Kernel Selection . . . 17
  - 2.3.2 Open-ended Kernel Generator . . . 18
  - 2.3.3 Generative Parameter Optimization . . . 22
  - 2.3.4 Optimizing Policy Distribution . . . 23
- 2.4 Empirical Study . . . 24
  - 2.4.1 Synthetic Kernel Recovery . . . 26
  - 2.4.2 Kernel Selection for Regression Tasks . . . 28
- 2.5 Conclusion . . . 30

3 Minimizer Sketch Design **31**
- 3.1 Introduction . . . 31
- 3.2 Problem setting . . . 34
  - 3.2.1 Notation . . . 34
  - 3.2.2 Other sketching variants . . . 36
  - 3.2.3 Other MSD approaches . . . 37
- 3.3 Density optimization of the minimizer scheme . . . 38
  - 3.3.1 Proxy Objective . . . 40
  - 3.3.2 Specification of TemplateNet . . . 42
  - 3.3.3 Specification of Distance Metric . . . 46
  - 3.3.4 Empirical study . . . 47
- 3.4 GSS optimization of the masked minimizer generalization . . . 58
  - 3.4.1 Generalized sketch score . . . 61

3.4.2 Masked minimizers . . . 61
3.4.3 Relating density and conservation metrics of masked minimizers . . . 62
3.4.4 Optimizing masked minimizers . . . 63
3.4.5 Empirical study . . . 66
3.5 Conclusion . . . 78

4 Federated Neural Architecture Search 81
4.1 Introduction . . . 81
4.2 Problem setting . . . 84
4.3 Related work . . . 85
4.3.1 Cell-based over-parameterized search space . . . 85
4.3.2 Scheduled dropout . . . 86
4.3.3 Continuous relaxation . . . 86
4.3.4 Federated NAS . . . 87
4.4 Methodology . . . 87
4.4.1 Cell-based Architecture Space . . . 88
4.4.2 Personalized Federated Learning Objective . . . 90
4.4.3 Context-Aware Operator Sampling . . . 93
4.5 Empirical Study . . . 94
4.5.1 Heterogeneous predictive tasks . . . 96
4.5.2 Tasks with varying heterogeneity levels . . . 97
4.5.3 Knowledge transfer to completely new tasks . . . 98
4.6 Conclusion . . . 99

5 Meta-Learning for Heterogeneous Tasks 101
5.1 Introduction . . . 101
5.2 Problem setting . . . 104
5.3 Related work . . . 105
5.3.1 Meta-learning . . . 105
5.3.2 Heterogeneous meta-learning . . . 106
5.3.3 Routing neural networks . . . 106
5.4 Methodology . . . 107
5.4.1 Gumbel-Beneš Routing Layer . . . 109
5.5 Empirical Study . . . 111
5.5.1 Meta-learning for uni-modal task distribution . . . 112
5.5.2 Meta-learning for multi-modal task distribution . . . 114
5.5.3 4x4 Jigsaw Mini-Imagenet . . . 116
5.5.4 Scalability of the modulation networks . . . 118
5.6 Conclusion . . . 119

6 Conclusion and Future Directions 121

Bibliography 127

List of Figures

2.1 The generic workflow of DTERGENS . . . 17
2.2 Schematic of the kernel generator . . . 20
2.3 Kernel recovery error . . . 27
2.4 Regression error for kernel selection . . . 28
2.5 Linear-periodic data pattern and number of unique kernels . . . 29

3.1 The PRIORITYNET architecture . . . 41
3.2 Visualization of PRIORITYNET and TEMPLATENET score assignments . . . 49
3.3 Best density factors obtained on HG38 and CHRXC . . . 50
3.4 Comparing performance of different distance metrics . . . 51
3.5 Comparing performance of different minimizer optimization methods on CHR1 and CHRX . . . 52
3.6 Comparing performance of different minimizer optimization methods on HG38 and CHRXC . . . 53
3.7 Comparing density and number of unique k-mers of different benchmarks on CHR1 54
3.8 Comparing density and number of unique k-mers of different benchmarks on CHRXC . . . 55
3.9 Comparing loss values and density factors of different template models . . . 56
3.10 Comparing density and runtime of DEEPMINIMIZER with different k values . . . 57
3.11 Comparing various sketching metrics using different training losses and masks v 68
3.12 Comparing GSS of different masked minimizer variants . . . 69
3.13 Comparing conservation and density of different masked minimizer variants . . . 70
3.14 Comparing GSS of different optimization methods with $w = 10, k = 10$. . . . 71
3.15 Comparing GSS of different optimization methods with $w = 15, k = 10$. . . . 72
3.16 GSS vs. no. 1-entries (BTR4) and offset position (homopolymer-rich synthetic sequence) . . . 76
3.17 GSS vs. number of 1-entries on the remaining bacterial genomes . . . 76
3.18 Finding the relative conservation exploit for open-syncmers . . . 77

4.1 Personalizable architecture search space . . . 90
4.2 Average classification accuracy of different NAS methods . . . 96

5.1 Beneš network architecture . . . 103
5.2 Overview of the MRM meta-routing framework . . . 108

5.3 Training loss and average test accuracy on OMNIGLOT and MINI-IMAGENET datasets 113
5.4 Average test accuracy of various baseline methods on JIGSAW-OMNIGLOT and JIGSAW-MINI-IMAGENET dataset 115
5.5 Average training loss of various baseline methods on JIGSAW-OMNIGLOT and JIGSAW-MINI-IMAGENET dataset 116
5.6 Average accuracy and training loss of various baseline methods on the 4×4 JIGSAW-MINI-IMAGENET dataset 117
5.7 Average training loss and test accuracy of various baseline methods on the multi-modal FLOWERS-AIRCRAFT-FUNGI dataset (sequential injection) 118
5.8 Average training loss and test accuracy of various baseline methods on the multi-modal FLOWERS-AIRCRAFT-FUNGI dataset (simultaneous injection) 119

List of Tables

3.1 Descriptions and lengths of benchmark sequences 67
3.2 Comparing GSS of gradient-based methods across 3 different training losses and 6 settings of (w, k) 73
3.3 Comparing GSS of non-gradient methods across 3 different training losses and 6 settings of (w, k) 74
3.4 Optimized masks found across different training losses and combinations of (w, k) 75

4.1 List of operators to be sampled at every edge of the master network 95
4.2 Predictive accuracy of CA-FEDPNAS FEDDSNAS on tasks with varying heterogeneity levels 98
4.3 Predictive accuracy on unseen tasks 99

5.1 Runtime per epoch and size of different modulation networks 120

Chapter 1

Introduction

Algorithms are typically developed with parameters that can be configured by users depending on their use cases. Although it is convenient to use a default configuration for every application, such a tactic is sub-optimal when the algorithm's performance is sensitive to the choice of configurations [5, 96]. For example, choosing an appropriate kernel function for kernel methods such as support vector machine [17] and Gaussian processes [83] to model data correlation can strongly influence the outcome of probabilistic inference [23]. In deep learning, the design of a neural network architecture (e.g., the number of layers, types of activation functions and the number of neurons in each layer) must be selected to achieve optimal performance on specific types of task [35, 80, 113].

In domains where performance variance is large, a more practical approach is therefore to calibrate the design of an algorithm on a *per-task* basis. That is, we seek to find the most suitable model configuration for each new problem instance. However, this calibration step has traditionally relied on the expertise of domain experts and heuristics, making it challenging to scale up due to the repetitive tuning effort involved. The lack of a principled and automated approach for configuring algorithmic solutions has motivated the study of automated algorithm design (AAD) through systematic optimization frameworks, which were first considered in the work of Rice [85] and subsequently in various algorithmic domains such as deep learning [35, 46, 80], non-

parametric Bayesian methods [23, 66, 69] and discrete algorithms [110, 111]. Formally, the general AAD problem is described as follows:

Definition 1 (Automated Algorithm Design) *Let $\mathcal{T}$ be an arbitrary set of computational tasks, and $\mathcal{M}$ be a likely infinite set of algorithms capable of solving any task $\tau \in \mathcal{T}$. Given a performance evaluation function $F : \mathcal{T} \times \mathcal{M} \to \mathbb{R}$, we say that an algorithm $m \in \mathcal{M}$ outperforms $m' \in \mathcal{M}$ on some task $\tau \in \mathcal{T}$ if and only if $F(\tau, m) > F(\tau, m')$. The AAD problem can be written as the following optimization task, which seeks to find the optimal algorithm $m_* \in \mathcal{M}$ such that $F(\tau, m')$ is maximized:*

$$m_* \quad \in \quad \underset{m \in \mathcal{M}}{\operatorname{argmax}} \, F(\tau, m) \, . \tag{1.1}$$

Due to the general nature of the performance measuring function F, the AAD task in Eq. (1.1) is commonly viewed as a black-box optimization (BBO) problem, where F is taken for an oracle that can be queried at will given some input configuration. Most existing black-box optimization methods tend to approach this task via a sequential optimization strategy that alternates between evaluating observations and making informed decision about subsequent probings of the oracle. Typically, this decision is either guided by practical heuristics or a surrogate model that estimates the relationship between configurations and evaluated performances. We summarize a few typical BBO methods in the following section.

1.1 Classical methods for automated algorithm design

1.1.1 Heuristic search

One of the most classical techniques to optimize a black-box function is grid search, which systematically evaluates all input configurations projected on a lattice to find the best performing candidate. For continuous configurations, this lattice is obtained by discretizing the search space, whereas for categorical configurations, this routine simply means exhaustively trying all combinations of values in each dimension. Alternatively, another widely-used approach for black-box

optimization is random search, where inputs are chosen completely at random to evaluate. Both grid search and random search are straight-forward to implement and have been used in several AAD tasks [10, 11, 62]. These methods, however, only focus on exploring the search space and have no built-in mechanism to exploit the collected observations, thus tend to scale poorly with the complexity of the input domain.

In contrast, other heuristic optimization methods such as coordinate ascent [31, 103] and simulated annealing [12] focus on refining the best candidate found so far in a greedy manner. Coordinate ascent achieves this by successively optimizing along a specific dimension while fixing every other dimension of the parameter space, repeating for all dimensions until convergence. Instead of making queries along a single coordinate, the simulated annealing algorithm alternatively evaluates a small neighborhood around the current best estimate and tries to move in the direction that yield the best improvement. Both methods have also been used in many AAD tasks [7, 11, 19], but are typically vulnerable to being trapped in local optima due to a lack of exploration mechanism.

1.1.2 Evolutionary strategies

Evolutionary strategies (ES) are optimization techniques inspired by nature, in which a population of configurations is set to evolve over time and improve its averaged performance while doing so. For every generation of this population, most evolutionary algorithms will conduct the following three steps: (1) estimate the fitness of each individual in the population; (2) generate new individuals using certain reproduction and/or mutation operators; and (3) replacing unfit individuals with new individuals.

Evolutionary algorithms [6, 30] have been widely applied in automated algorithm design, such as model selection for deep neural networks [61, 65, 75, 78] and support vector machines (SVM) [63]; or finding clinical interventions [76]. These techniques fundamentally differ from the above heuristic algorithms by balancing between exploration and exploitation. For example,

in genetic optimization, high-fitness individuals can partially pass down their representations to the next generation via a crossover operator (i.e., exploitation), in hope that good performances can be preserved and improved. At the same time, new representations are continually generated via the random mutation operator (i.e., exploration), thus ensuring the optimization is not trapped in local optima.

1.1.3 Sequential model-based optimization

Unlike the above methods which are considered model-free, sequential model-based optimization (SMBO) is an optimization paradigm which iteratively uses collected information to estimate a surrogate model for the black-box function F. This model is then used to guide the acquisition of subsequent observations. Different SMBO methods have been used to address various AAD tasks such as configuring parameterized tree search algorithms and local SAT solvers [45]. We now give a description of the Bayesian optimization (BO) algorithm [93], which is a widely-used variant of SMBO.

The general BO algorithm usually prescribes a Gaussian Process (GP) prior [83] over the black-box objective function, i.e. $F \sim \mathcal{GP}(\mu, k)$ where $\mu : \mathcal{M} \to \mathbb{R}$ and $k : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ are respectively the prior GP mean and covariance functions. This prior implies that for any finite subset of candidate configurations $\{m_1, m_2, \ldots, m_T\}$ the corresponding performance vector $[F(m_1), F(m_2), \ldots, F(m_T)]$ is normally distributed *a priori* with mean $[\mu(m_1), \mu(m_2), \ldots, \mu(m_T)]$ and covariance $[k(m_i, m_j)]_{i,j \in [T]}$. At any iteration t, the BO algorithm then uses this prior distribution and the set $\mathcal{D}_t$ of collected observations so far to derive a posterior predictive distribution $p(F(m_*) \mid m_*, \mathcal{D}_t)$ for any subsequent candidate m_*.

The posterior predictive mean can be used directly to estimate the expected performance of subsequent candidates, thus allowing us to exploit high-performing configurations without actually evaluating F. However, since the posterior naturally has high uncertainty in unobserved regions, doing so would discourage thorough exploration of the search space and risk missing

out on good solutions. To address this issue, the BO algorithm instead constructs an *acquisition function* that incorporates both the posterior mean and covariance to balance this exploitation and exploration trade-off. For example, the upper confidence bound (UCB) acquisition function proposed by Srinivas et al. [94] is given by:

$$\alpha_{\text{UCB}}(m; M_t, \sigma_t) \triangleq M_t(m) + \beta\sqrt{\sigma_t(m)}\,, \tag{1.2}$$

where $M_t(m)$ and $\sigma_t(m)$ respectively denote the posterior mean and variance at iteration t given some candidate m. Here, the parameter β reflects the trade-off between exploiting candidates with high expected performance and exploring candidates with high uncertainty. Finally, the BO algorithm can be described via the following update rules:

$$\begin{aligned} m_{t+1} &= \arg\max_{m\in\mathcal{M}} \alpha_{\text{UCB}}\left(m; M_t, \sigma_t\right)\,, \\ \mathcal{D}_{t+1} &= \mathcal{D}_t \cup \{(m_{t+1}, F(m_{t+1}))\}\,, \\ M_{t+1}, \sigma_{t+1} &\leftarrow p\left(F(m_*) \mid m_*, \mathcal{D}_{t+1}\right)\,. \end{aligned} \tag{1.3}$$

Nonetheless, we remark that the vanilla BO algorithm is best suited for low-dimensional and continuous input domains. In practical AAD tasks where the space of configuration is structured and discrete, there is generally no unifying approach to parameterize the mean and covariance function of the GP surrogate. Furthermore, optimizing the acquisition functions will also become non-trivial and require special modelling considerations.

1.2 Domain-specific AAD Problems

In many specific AAD problems, $\mathcal{M}$ can be specified as a sub-class of algorithms that is described by discrete data structures such as sub-trees [23], graphs [8, 35, 80] or permutations [70]. This choice of representation implicitly prescribes a correlation structure among candidate models via the topology of $\mathcal{M}$ and thus allows the general AAD objective to subsequently be cast as optimization/search tasks on these structured domains.

This thesis will focus on several prototypical classes of AAD tasks that can be unified through the lens of structured optimization, namely kernel selection (KS), minimizer sketch design (MSD), and neural architecture search (NAS). In particular, the KS problem in Bayesian statistics [23, 69] can be formulated as a tree search routine to find the composite function that best models the covariance structure of a stochastic process. The MSD problem, which seeks to find an optimal sketching scheme for biological sequences, is expressed as finding the optimal permutation of all fixed length substrings induced by the sequence vocabulary [70, 110, 111]. Finally, the NAS problem is approached via setting $\mathcal{M}$ to be a set of computation graphs that share the same vertex and edge space, which respectively denotes all intermediate feature representations and all possible transformations [80, 113].

Even though this domain restriction strategy has enabled more meaningful formulations of the AAD problem, their resulting discrete/combinatorial objectives are still challenging to solve due to the prohibitive sizes of their search domains. For example, without any further restriction, the tree search space in the KS problem naturally has unbounded depth, whereas the graph and permutation domains of the MSD/NAS problems are virtually infinite in most practical settings. In practice, this large amount of admissible candidate configurations would render the majority of standard approaches such as integer programming [3, 4] and heuristic search inefficient, and thus necessitates better informed search strategies to navigate these massive search spaces.

Another major challenge of AAD arises due to the fact that none of the respective performance measuring functions have a closed-form expression, nor are they computationally cheap to evaluate. For example, measuring the fitness of a neural network architecture would typically entail training all layer parameters to convergence before measuring its predictive loss/accuracy with respect to some validation dataset. As such, many strategies that rely on repeatedly probing this evaluation function are inefficient and not applicable in practice.

To address these challenges, a significant portion of the existing AAD literature is centered around the idea of developing an efficient knowledge transfer mechanism that can generalize

algorithmic behaviors across different configuration instances. In particular, methods under this paradigm often fall under two broad categories. The first category includes methods that aim to infer unseen regions of high-performing configurations, such as sequential model-based optimization approaches that iteratively refine a surrogate performance function and leverage this to guide the acquisition of new observations [51, 66]. The second category includes methods that aim to reduce the cost of evaluating new candidates, focusing on making structural assumptions about the search space to facilitate reusing knowledge from past evaluations. An example of such method is the weight sharing scheme in neural architecture search [35, 80], where every candidate architecture is constructed from the same pool of building-block layers. Each layer in this search space is continually optimized throughout all search iterations (i.e., whenever selected by the search algorithm), thus will alleviate the need to re-train each architecture from scratch. We will defer the review of these domain-specific methods to their corresponding chapters.

Many approaches in the various AAD domains that we investigate (i.e., KS, MSD and NAS) either suffer from the scalability issues above, or must rely on additional domain restrictions to sufficiently prune their massive search spaces. For instance, existing KS methods [23, 66, 69] typically require that all candidate kernel functions must have upper-bounded complexities (e.g., functions with short expressions), whereas the permutation domain in the MSD problem is predominantly approximated by the space of sparse hitting sets [26, 110]. In practice, while such assumptions serve to ensure the feasibility of their respective optimization objectives, they are typically heuristics that do not factor in the optimality of the pruned candidates.

We note that most AAD frameworks only focus on exploiting the observed performances of configurations in the same task domain. In practice, this is not always the only source of information to guide exploration of the candidate space. For instance, in applications that require tuning for many related tasks, it would be more intuitive to leverage the combined knowledge from all task domains, rather than to independently approach these problems with their respective observations. Motivated by these shortcomings, this thesis therefore seeks to address the follow-

ing research question: Can we design information-efficient frameworks that can better exploit both **intra-task** and **inter-task** information, and thus improve the scalability and performance for automated algorithm design?

1.3 Thesis aims and contributions

To address the research question above, this thesis aims to exploit intra-task information through learning complementary components that can implicitly reason about the desirability of candidate configurations. This will improve the efficiency of AAD without having to fully commit to expensive performance evaluations. In this direction, we ground our study in two specific families of AAD instances, namely the kernel selection (KS) problem, and the minimizer sketch design (MSD) problem.

Chapter 2 investigates a novel search framework for composite kernel functions through learning a generative component. We show that the accuracy and efficiency of kernel performance estimation can be improved through learning an early stopping policy that optimally truncates infinitely long trajectories of expressions. On the other hand, Chapter 3 studies the design problem of various string sketching/compression methods. We similarly develop an innate reasoning component that takes the form of a template model generating desirable substring ranking patterns. Learning this template model allows us to design surrogate objective functions (for originally challenging permutation learning tasks) that can be efficiently optimized. In both settings, we show that our proposed learning objectives are more effective formulations of their respective AAD objectives, and that they yield better performing configurations than other state-of-the-art methods.

Orthogonal to the above direction, the second aim of this thesis is to develop *multi-task* knowledge transfer paradigms that leverage information sharing across *different tasks* to improve the performance and efficiency of concurrent AAD instances. Specifically, Chapter 4 studies an extension of the neural architecture search problem in the federated learning setting [72], where

an ensemble of predictive tasks collaborate to find optimal architectures for their respective computational goals. Chapter 5 further investigates a new meta-learning approach [28] that can effectively leverage existing model configuration experiences to address unseen and heterogeneous AAD instances.

The remainder of this thesis will be organized as follows:

- Chapter 2 investigates the kernel selection (KS) problem, which restricts the candidate domain to an infinite-depth tree induced by the kernel composition language [23]. Our contribution is a novel Bayesian optimization framework that optimizes a recurrent kernel generator that outputs infinitely long sequences of kernel expressions. We jointly learn a termination policy model which decides the optimal stopping point along each infinite generative trajectory. We show that this strategy results in better performing kernel functions on a wide range of predictive tasks.
- Chapter 3 studies the minimizer sketch design (MSD) problem, which aims to find a permutation of substrings that induces the optimal sketch of a sequence via the minimizer algorithm [70, 89]. Our contribution is a novel optimization framework, called DEEPMINIMIZER, which formulates as a pair of substring ranking networks which guarantee different properties of a good solution. Through negotiating for a consensus outcome, these networks result in a continuous relaxation of the original discrete permutation learning objective. We show that our method significantly improves the state-of-the-art performance and scalability of MSD. We further extend our algorithm to unify and provide sketch designs for other sequence sketching methods, such as the syncmer approach [24, 90].
- Chapter 4 investigates the neural architecture search (NAS) problem in a federated learning setting [72], which seeks to concurrently optimize the solution architectures for multiple related tasks without exposing their private data. Our contribution is a novel personalized learning objective which distills the aggregated problem solving experiences into hierarchical architecture components which can be fine-tuned to solve each specific task. We show

that this knowledge sharing scheme is capable of maintaining a high degree of personalization in the solution ensemble and thus significantly improves the efficiency of multi-task neural architecture search.

- Chapter 5 extends the *multi-task* AAD paradigm to deal with unseen and heterogeneous tasks. Specifically, drawing inspiration from the meta-learning paradigm for multi-task scenarios, we aim to learn a common meta-configuration that can be adapted quickly to solve any specific task. Unlike the standard meta-learning setting, we consider the scenario where the AAD task distribution might be highly heterogeneous, and hence there might not be a single configuration that can adapt to every task. To address this heterogeneous meta learning (HML) problem, we implicitly model the multi-modality of the task distribution via an adaptive neural routing system within the network architecture. This motivates the development of a parameter-efficient channel shuffling module for convolutional architectures that can be meta-learned via an extension of the MAML training objective [28]. We show that, on a wide range of multi-modal meta-learning benchmarks, our new heterogeneous meta learning framework outperforms previous methods in both generalization accuracy and convergence speed.
- Chapter 6 summarizes related works beyond the scope of this thesis that are published during my PhD journey. We also discuss the impact of this thesis and its various potential future directions.

Chapter 2

Kernel Selection

2.1 Introduction

In many traditional machine learning algorithms, working with high-dimensional feature spaces can be computationally expensive or even infeasible. To address this, many existing methods use the kernel trick to implicitly map data points into high-dimensional feature spaces without explicitly computing the transformations. The kernel trick works by defining a kernel function that models the similarity or dissimilarity between pairs of data points in the original space. This kernel function computes the dot product between the transformed feature vectors in some latent high-dimensional space, without explicitly carrying out the costly transformations. By doing so, it avoids the need to store or compute the explicit feature representations, effectively sidestepping the computational burden associated with high dimensions.

With many kernel functions available, ranging from Gaussian radial basis functions to polynomial and sinusoidal kernels, the choice of an appropriate kernel becomes a crucial step in building powerful and effective machine learning models. Kernel selection is far from a one-size-fits-all approach. The decision of which kernel to employ depends on numerous factors, including the characteristics of the dataset, the nature of the problem at hand, and the desired trade-offs between accuracy, computational efficiency, and interpretability. Moreover, different

kernels possess unique properties and capture distinct types of patterns, making the selection process an intricate task that requires domain knowledge, experience, and a significant amount of trial-and-error effort.

Various strategies have been proposed to automate this selection problem. Duvenaud et al. [23] formulates the kernel selection problem as constructing composite kernels from atomic kernel functions, and hence casts it as a tree search problem guided by the model likelihood score. However, this requires fixing the kernel parameters during the tree search, and training them after the search concludes. The trained parameters are used as initialization for the next round of tree search, repeated until convergence or a sufficiently good kernel function is found. Nonetheless, the parameters optimized with respect to one kernel function might not induce similar predictive behaviors on another function, thus it is inconclusive whether the model likelihood heuristic accurately estimates the kernel performance.

Malkomes et al. [69] employs a variant of the Bayesian optimization (BO) algorithm [93] that models the kernel covariance function of the GP surrogate using the Hellinger distance. While this method partially alleviates the initialization problem from Duvenaud et al. [23] by sampling kernel parameters from prior distributions to estimate the proposed Hellinger kernel, it cannot tractably update these priors as the BO algorithm acquires more observations. To address this problem, Lu et al. [66] first trains a variational autoencoder (VAE) model [53] to acquire a latent embedding space of kernel functions. Lu et al. [66] then employs BO to optimize for a latent representation that decodes into the optimal kernel function given the task data, thus bypassing the challenge of modelling a kernel function on discrete objects.

We note that all of the above methods commonly require further restrictions of the candidate space to a finite set to ensure feasibility. Particularly, Duvenaud et al. [23] assumes a finite-depth tree such that the tree search algorithm can successfully terminate. The BO approach of Malkomes et al. [69] does not have a scalable solution for optimizing its acquisition function and has to confine the search space to a small number of active candidates, such that their

acquisition values can be exhaustively computed. Similar to Duvenaud et al. [23], the VAE parameterization of Lu et al. [66] also assumes that the length of any decoded kernel expression is upper-bounded.

To address the various shortcomings above, this chapter proposes a more expressive reformulation that does not require any additional domain restriction. Our framework, Dynamic Termination Generative Search (DTERGENS), casts the kernel selection problem as optimizing the weights of a generative model, whose mechanism explicitly mirrors the kernel composition rules via a recurrent formulation. Unlike Lu et al. [66], which treats each kernel expression as an atomic instance to be embedded on a continuous latent space, our approach instead maps an *entire subspace* of kernel expressions to each latent representation. Explicitly, this is achieved via formulating our generative model as an open-ended process which synthesizes an infinite trajectory of kernel expressions given a unique weight initialization. Our AAD objective can be seen as finding a subspace that contains the optimal model, rather than searching for the optimal model itself. This relaxation subsequently allows us to capture an unrestricted and significantly more expansive set of candidate kernel functions, compared to existing approaches such as Duvenaud et al. [23], Lu et al. [66], Malkomes et al. [69].

Nonetheless, infinitely complex kernel expressions are generally not meaningful and are impractical to compute. In addition to our relaxed objective, we further seek to distill the optimal kernel function from the optimal trajectory. To this end, we introduce a data-driven policy that learns to optimally terminate the generative model to maximize performance. Intuitively, the interplay between the generator model and this policy component can be interpreted as a two-step decision making process, in which the generator weight determines a subspace of candidates, and the policy learns to select candidates in this generative trajectory. Last, we introduce a novel bi-level Bayesian optimization scheme to jointly optimize both components.

We demonstrate that our method is able to produce complex kernels which significantly improve predictive performance of multiple predictive tasks over state-of-the-art structure search

methods. Our results show a wider range of structures being explored by DTERGENS and more rapid rates of improvement as compared to other methods. Finally, we show that DTERGENS is also able to recover known well-performing kernels on artificially designed predictive tasks. This chapter is co-authored by Carl Kingsford and was published at the International Conference for Machine Learning (ICML) in 2020 [38].

2.2 Problem setting

We ground our study in the context of the multivariate regression problem $y = g(\mathbf{x}) + \epsilon$ given real-valued observation tuples $\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^{d+1}\}_{i \in [N]}$, where d is the input dimension, ϵ denotes the observation noise and g is the latent function to be inferred. Typically, the prior distribution of g will be modelled such that a tractable posterior predictive distribution $p(g(\mathbf{x}_*) \mid \mathbf{x}_*, \mathcal{D})$ can be analytically derived. For example, in Gaussian process (GP) regression [83], if g is distributed by a GP prior [83] with zero mean and covariance function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, such that for any finite subset of inputs $\{\mathbf{x}_{i_1}, \mathbf{x}_{i_2} \ldots \mathbf{x}_{i_m}\}$ where $i_1, i_2 \ldots i_m \in [N]$, we have:

$$\begin{bmatrix} g(\mathbf{x}_{i_1}) \\ g(\mathbf{x}_{i_2}) \\ \ldots \\ g(\mathbf{x}_{i_m}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ \ldots \\ 0 \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_{i_1}, \mathbf{x}_{i_1}) & k(\mathbf{x}_{i_1}, \mathbf{x}_{i_2}) & \ldots & k(\mathbf{x}_{i_1}, \mathbf{x}_{i_m}) \\ k(\mathbf{x}_{i_2}, \mathbf{x}_{i_1}) & k(\mathbf{x}_{i_2}, \mathbf{x}_{i_2}) & \ldots & k(\mathbf{x}_{i_2}, \mathbf{x}_{i_m}) \\ \ldots & \ldots & \ldots & \ldots \\ k(\mathbf{x}_{i_m}, \mathbf{x}_{i_1}) & k(\mathbf{x}_{i_m}, \mathbf{x}_{i_2}) & \ldots & k(\mathbf{x}_{i_m}, \mathbf{x}_{i_m}) \end{bmatrix} \right).$$

If the observation noise ϵ is Gaussian, then the posterior distribution $p(g(\mathbf{x}_*) \mid \mathbf{x}_*, \mathcal{D})$ is also Gaussian and can be tractably derived. This modelling choice of the covariance matrix, broadly referred to as the *kernel trick* [43], is widely used in the GP literature as well as many other probabilistic methods to provide non-linear transformations of data onto some high-dimensional feature space. However, since covariance matrices are required to be positive semi-definite, not all functions will suffice as a *kernel function* in the above formulation. The formal condition for a kernel function k to be valid is given as follows:

Definition 2 (Valid Kernel) *A kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is valid if and only if for any subset of inputs $\{\mathbf{x}_{i1}, \mathbf{x}_{i2} \dots \mathbf{x}_{im}\}$ and for all $\gamma \in \mathbb{R}^m$ we have:*

$$\sum_{u=1}^{m} \sum_{v=1}^{m} \gamma_u \gamma_v \cdot k(\mathbf{x}_{iu}, \mathbf{x}_{iv}) \quad \geq \quad 0 \,, \tag{2.1}$$

where γ_u and γ_v respectively denotes the u^{th} and v^{th} entry of γ.

Although there are infinitely many constructions that satisfy the above condition, we can derive a systematic method to derive kernel functions via procedurally combining simple *base* kernels to generate more sophisticated *composite* kernels. As formalized in Duvenaud et al. [23], a popular set of kernel composition rules can be described as follows:

- Let $\mathcal{M}$ be the set of all valid kernel functions, we first select $\mathcal{K}_{\mathcal{B}} \subset \mathcal{M}$ as a base kernel set. We say that $\mathcal{K}_{\mathcal{B}}$ induces a set of composite kernels $\mathcal{K}_{\mathcal{C}}$, such that $\mathcal{K}_{\mathcal{B}} \subset \mathcal{K}_{\mathcal{C}} \subseteq \mathcal{M}$. The membership of $\mathcal{K}_{\mathcal{C}}$ is recursively defined via the following rules:
- Addition: if $k_1, k_2 \in \mathcal{K}_{\mathcal{C}}$ and $\forall \mathbf{x}, \mathbf{x}' : k_+(\mathbf{x}, \mathbf{x}') \triangleq k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$, then $k_+ \in \mathcal{K}_{\mathcal{C}}$.
- Multiplication: if $k_1, k_2 \in \mathcal{K}_{\mathcal{C}}$ and $\forall \mathbf{x}, \mathbf{x}' : k_+(\mathbf{x}, \mathbf{x}') \triangleq k_1(\mathbf{x}, \mathbf{x}') \times k_2(\mathbf{x}, \mathbf{x}')$, then $k_\times \in \mathcal{K}_{\mathcal{C}}$.

For ease of notation, we will drop the input argument and instead treat kernel functions as basic operands (e.g., $k_+ = k_1 + k_2$ or $k_\times = k_1 \times k_2$) when referring to these composition rules. We further remark that the notation $k \in \mathcal{K}_{\mathcal{C}}$ only implies the *discrete form* of the kernel function, which does not include its learnable parameters. For example, consider the general squared exponential (SE) kernel $k_{\text{SE}} \in \mathcal{K}_B$ such that $k_{\text{SE}}(\mathbf{x}, \mathbf{x}') \triangleq \sigma^2 \exp\left(\sum_{t=1}^{d} \ell_t^{-2} \cdot (\mathbf{x}_t - \mathbf{x}'_t)^2\right)$ is parameterized by the signal variable σ and the length-scale variables $\boldsymbol{\ell} = \{\ell_t\}_{t=1}^{d}$. We do not count different initializations of k_{SE} in $\mathcal{K}_{\mathcal{C}}$ because the maximum likelihood estimation (MLE) of these parameters is well-studied and can be implicitly incorporated into the performance metric. Finally, the kernel selection problem can be formalized as follows:

Definition 3 (Kernel Selection) *Let $\tau \triangleq \{\mathcal{K}_{\mathcal{B}}, \mathcal{D}, \mathcal{A}\}$ describe a kernel selection task where $\mathcal{K}_{\mathcal{B}}$ is the set of base kernels; $\mathcal{D}$ is the set of provided observations (i.e., train/validation/test data); and $\mathcal{A}$ denotes a kernel-based algorithm specified by some $k \in \mathcal{K}_{\mathcal{C}}$ that is induced by $\mathcal{K}_{\mathcal{B}}$. We*

further let $F_\tau : \mathcal{K}_\mathcal{C} \to \mathbb{R}$ be the performance measuring function of this task, which (1) executes a well-defined algorithmic procedure to optimize the continuous parameters of some candidate kernel function k (e.g., applying gradient descent update with respect to the MLE objective until convergence) and (2) returns the predictive accuracy evaluated on the test set. Then, the kernel selection problem is succinctly stated as:

$$k_* \quad = \quad \underset{k \in \mathcal{K}_\mathcal{C}}{\operatorname{argmax}} \, F_\tau(k) \; , \tag{2.2}$$

where k_* denotes the optimally performing kernel function with respect to τ.

2.3 Methodology

Similar to Lu et al. [66], we reformulate the discrete optimization problem as a BO task over a latent representation space, which allows us to bypass both the need to select an initial set of active candidates [23] and to rely on heuristic methods for exploring new candidates [69]. However, instead of learning a direct embedding, we implicitly encode composite kernels as output of a parameterized *open-ended recurrent generator*. While Lu et al. [66] focuses on learning a mapping between latent representations and kernel expressions, our method, DTERGENS, learns a mapping between the latent coordinates and the *infinitely long* kernel expression generative trajectories. This main difference helps to avoid placing an explicit upper limit on the expression length, thus ensuring sufficient expressiveness of the candidate set.

The VAE decoder in [66], which sequentially generates the next operand and operator given the current expression, is pre-trained with the VAE encoder using randomly sampled kernel expressions. In contrast, our generator explicitly captures the *sum-of-product* structure through a nested recursion procedure, thus is capable of generalizing the composition rules to create new expressions. To prevent the generation of infinitely long and computationally expensive expressions, we learn a stochastic early stopping policy, which determines the best performing stopping point on any generative trajectory. This method enables the exploration of arbitrarily complex

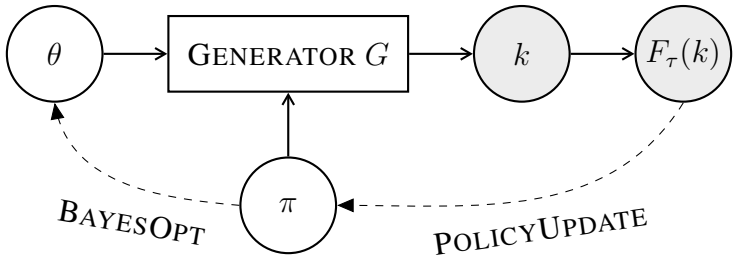


Figure 2.1: The generic workflow of DTERGENS. Given policy π, we employ BO to obtain generative weight candidate θ (Section 2.3.3). Using the observed generative trajectory, we alternately update the policy distribution (Section 2.3.4).

expressions and is the first selection method that places no further structural restriction on the search space.

Both the generative parameters and the termination policy can be jointly optimized by exploiting their dynamic in the generative component. Fixing a termination policy, we devise a dynamic BO algorithm for optimizing generative parameters that is capable of adapting to the constant policy updates. Alternately, given each sample trajectory collected by the BO step, we devise an update algorithm for the policy distribution via modelling the dynamic between these two components. Together, these steps compose our main contribution, which is the DTERGENS algorithm for composite kernel selection.

We demonstrate that DTERGENS is able to produce sophisticated kernel expressions, which significantly improve the predictive performance over state-of-the-art methods on multiple benchmarks. Our results show a wider range of structures being explored by DTERGENS and faster convergence compared to other methods. Finally, we show that DTERGENS is also able to recover known well-performing kernels on artificially designed predictive tasks.

2.3.1 Reformulating Kernel Selection

Directly optimizing over the discrete domain of kernel expressions as suggested by Definition 3 is challenging. To work around this, we model k as the output of a generative process $G(\theta, \pi)$

conditioned on generative weights $\theta \in \Theta$ and a policy $\pi \in \Pi$ that controls the termination of G. The designs of G and π are given in subsequent sections. We approximate the kernel selection objective in Definition 3 as:

$$\arg\max_{k \in \mathcal{K}_{\mathcal{C}}} F_\tau(k) \quad \simeq \quad \arg\max_{\theta \in \Theta, \pi \in \Pi} R_\tau(\theta, \pi) \ , \tag{2.3}$$

where $R_\tau \triangleq F_\tau \circ G$ is the composition of the generative model G and the evaluation function F_τ.

We alternately optimize π and θ while fixing the other component. Explicitly, given a policy π, the generative parameter θ is optimized using an adapted BO algorithm that is formulated with the conditional policy distribution $p(\pi \mid \theta)$ (Section 2.3.3). On the other hand, given the BO-sampled observations, the policy distribution $p(\pi \mid \theta)$ can be updated via MLE (Section 2.3.4). The outline of this workflow is illustrated in Fig. 2.1. To lay the groundwork for our algorithmic development, we will first discuss the design of our kernel generator $G(\theta, \pi)$ in Section 2.3.2.

2.3.2 Open-ended Kernel Generator

We note that a composite kernel can be expressed as a sum-of-products over some base kernel units. That is, for any composite kernel $k \in \mathcal{K}_{\mathcal{C}}$, there exists a finite collection of base kernels $\{k_{t,t'}\} \subseteq \mathcal{K}_{\mathcal{B}}$ such that:

$$k \quad = \quad \sum_{t=1}^{m} \prod_{t'=1}^{n_t} k_{t,t'} \ . \tag{2.4}$$

Composite kernel expressions thus naturally manifest as tree structures with (1) a primary linear chain; and (2) several secondary linear branches that are connected to the primary chain. In this view, the secondary branches correspond to different products of base kernel units, whereas the primary chain corresponds the summation over these products. To generate such structures, we construct G by composing two nested recurrent units, which are described below and visualized in Fig. 2.2.

Generator Overview

Our generator architecture comprises a primary unit and a secondary unit that respectively generate the primary chain and the secondary branches. Each unit is parameterized by a recurrent neural network (RNN) and a termination policy. This policy predicts a stopping signal for the generative process given the hidden state of the neural network.

Every recurring step of the primary unit initiates a new secondary branch and computes an initial hidden state for the secondary RNN. On the other hand, every recurring step of the secondary unit generates a new base kernel unit on the current branch. Each unit will recur until its respective policy outputs a stopping signal given the respective current hidden state. When the primary unit terminates, we output a tree structure corresponding to a composite kernel. We give the mechanism of each component below.

Primary Unit

The primary unit $\mathcal{U}_p : \mathbb{R}^{d_p} \to \mathcal{K}_\mathcal{C}$ is given by the RNN G_p and the policy $\pi_p : \mathcal{K}_\mathcal{C} \to \mathbb{R}$. G_p has hidden dimension d_p and emits an initial hidden state for G_s. Given an arbitrary initial hidden state $h_0 \in \mathbb{R}^{h_p}$, G_p performs the following at any generative step $t \geq 0$:

- If $t = 0$, initialize candidate kernel expression $\bar{k} = 0$ (i.e., a constant function).
- Generate the next hidden state and the current emission output: $(h_{t+1}, h_{t,0}) \leftarrow G_p(h_t; \theta_t)$.
- Generate and append the t^{th} secondary branch to the current expression: $\bar{k} \leftarrow \bar{k} + \mathcal{U}_s(h_{t,0}; \theta_s, \pi_s)$.
- Query termination probability with the current intermediate expression: $\alpha \leftarrow \pi_p(\bar{k})$.
- With probability α, end the generative procedure and return $\bar{k}$ as a terminal expression.
- With probability $1 - \alpha$, set $t \leftarrow t + 1$ and repeat the procedure.

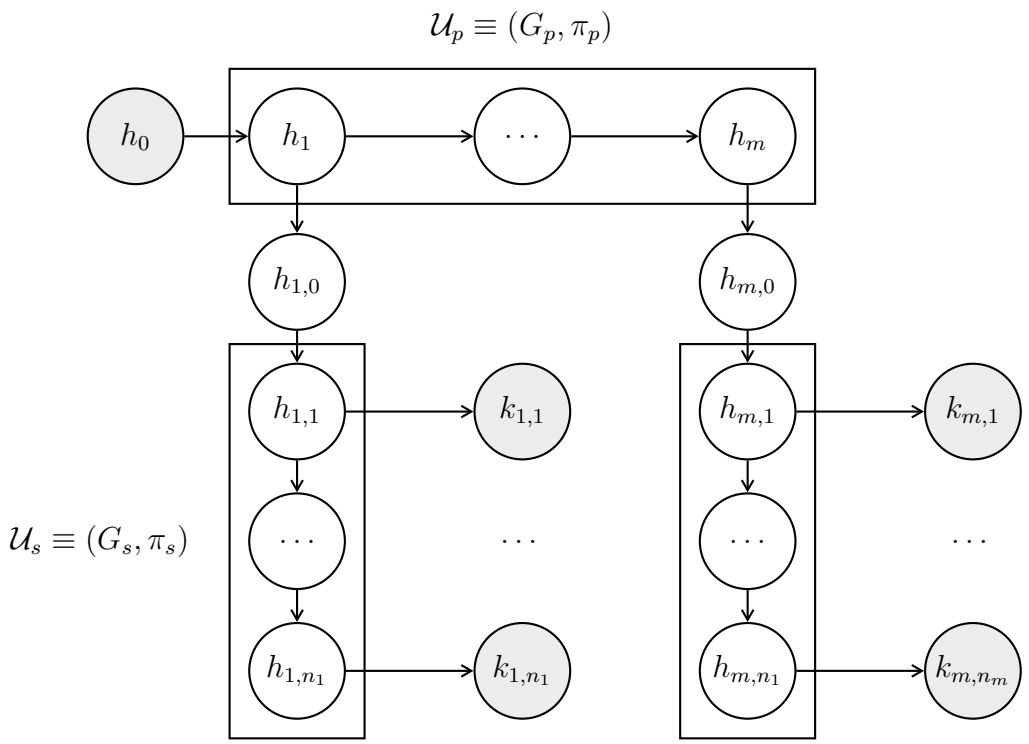


Figure 2.2: Schematic of the kernel generator with nested units $\mathcal{U}_p$ and $\mathcal{U}_s$. Each component recursively computes its next hidden state and emission output using respective recurrent neural network G_s and G_p. The termination probability at each generative step is determined by policies π_p and π_s. The final candidate kernel expression is composed using Eq. (2.4).

Secondary Unit

The secondary unit $\mathcal{U}_s : \mathbb{R}^{d_s} \to \mathcal{K}_\mathcal{C}$ is given by the RNN G_s and the policy $\pi_s : \mathcal{K}_\mathcal{C} \to \mathbb{R}$. G_s has hidden dimension d_s and emits one-hot representations of the base kernel units in $\mathcal{K}_\mathcal{B}$. Given an initial hidden state $h \in \mathbb{R}^{d_s}$ output by G_p, $\mathcal{U}_s$ generates a corresponding product of base kernel units. Given an initial hidden state $h_{t,0}$ produced by G_p at time t, G_s then performs the following at any inner loop generative step $t' \geq 0$:

- If $t' = 0$, initialize the inner loop kernel expression $\bar{k}_t = 1$ (i.e., a constant function).
- Generate the next hidden state and emit the current base kernel: $(h_{t,t'+1}, k_{t,t'}) \leftarrow G_s(h_{t,t'}; \theta_s)$.
- Extend current expression via multiplication: $\bar{k}_t = \bar{k}_t \times k_{t,t'}$.

- Query the termination probability $\beta \leftarrow \pi_s(\bar{k} + \bar{k}_t)$.
- With probability β, end the generative procedure for the current branch and return $\bar{k}_t$.
- With probability $1 - \beta$, set $t' \leftarrow t' + 1$ and repeat the procedure.

Termination Policy

Let $\tau = \{\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_n\}$ denote the set of training inputs specified by the learning instance Ω, as introduced in Definition 3. We respectively parameterize the termination policies π_p and π_s by two neural networks γ_p and γ_s as follows:

$$\begin{aligned} \pi_p(k; \gamma_p, \tau) &\triangleq \sigma\left(\sum_{i,j \in [n]} \gamma_p(\mathbf{x}_i) \cdot \gamma_p(\mathbf{x}_j) \cdot k(\mathbf{x}_i, \mathbf{x}_j)\right), \\ \text{and} \quad \pi_s(k; \gamma_s, \tau) &\triangleq \sigma\left(\sum_{i,j \in [n]} \gamma_s(\mathbf{x}_i) \cdot \gamma_s(\mathbf{x}_j) \cdot k(\mathbf{x}_i, \mathbf{x}_j)\right), \end{aligned} \tag{2.5}$$

where $\sigma(t) \triangleq 1/(1 + \exp(-t))$ denotes the sigmoid activation function. For convenience, the notation γ_p and γ_s are also used interchangeably to denote the weights of these neural networks

This data-driven parameterization serves to model the task-specific termination rules conditioned on observation task data, and accounts for the fact that different expression lengths are required for different tasks. We then model the interaction between the generative weights θ and the termination policy π via parameterizing the conditional distributions $p(\gamma_s \mid \theta)$ and $p(\gamma_p \mid \theta)$ with neural networks. These distributions are in turn used to construct a covariance function that measures similarity among candidate composite kernels. Using this covariance function, we introduce an adapted BO routine that alternates between generative weight optimization via policy sampling (Section 2.3.3), and policy update given collected trajectories of generated base kernels (Section 2.3.4).

2.3.3 Generative Parameter Optimization

This section details the BO step to optimize the generative weight θ. Formally, fixing a policy distribution $\bar{\pi}$ whose parameterization $\bar{\gamma}$ follows the conditional distribution $\bar{p}(\gamma \mid \theta) \triangleq \bar{p}(\gamma_p \mid \theta)\bar{p}(\gamma_s \mid \theta)$, the partial objective can be rewritten as:

$$\theta^* \;=\; \arg\max_{\theta\in\Theta} g_{\bar{\pi}}(\theta) \quad\equiv\quad \arg\max_{\theta\in\Theta} \mathbb{E}_{\gamma\sim\bar{p}}\left[R_\tau(\theta, \pi(\gamma))\right] \;. \tag{2.6}$$

We adopt the standard practice of BO [93] and impose a Gaussian Process (GP) [83] prior on the black-box function $g_{\bar{\pi}}$, i.e., $g_{\bar{\pi}} \sim \mathcal{GP}(\mu, k_{\text{BO}})$ where μ and k_{BO} respectively denote its mean and covariance functions. The BO algorithm obtains the next best candidate θ by maximizing a surrogate acquisition function constructed from the posterior distribution of this GP. The performance evaluation of the composite kernel generated by θ and π, $g_\pi(\theta)$ is then used to update the GP posterior.

Note that in standard BO setting, the functional landscape is static, whereas in our formulation the policy $\bar{\pi}$ updates after every BO iteration. To account for this dynamic update of $\bar{\pi}$, we will therefore model the GP covariance using two components: (1) an intrinsic kernel component that characterizes the feature distance between the generative weights θ_i and θ_j; and (2) an extrinsic kernel component that captures the divergence of their conditional policy distribution $p(\gamma \mid \theta_i)$ and $p(\gamma \mid \theta_j)$ given the current parameterization γ_p, γ_s. Explicitly, given candidates θ_i and θ_j, the kernel distance between these candidates is given as:

$$k_{\text{BO}}(\theta_i, \theta_j) \;\triangleq\; k_{\textsc{Policy}}(\theta_i, \theta_j) \cdot k_{\textsc{Generator}}(\theta_i, \theta_j) \;, \tag{2.7}$$

where $k_{\textsc{Generator}}$ is given by the standard squared exponential kernel and $k_{\textsc{Policy}}$ is given by the symmetric KL divergence between the policy distributions conditioned on θ_i and θ_j respectively:

$$k_{\textsc{Policy}}(\theta_i, \theta_j) \;\triangleq\; \mathbb{KL}\Big(p(\gamma \mid \theta_i) \parallel p(\gamma \mid \theta_j)\Big) \;. \tag{2.8}$$

2.3.4 Optimizing Policy Distribution

This section then details an update iteration of π given a new candidate weight θ (derived from maximizing the BO acquisition function) and its generated kernel expression $k = \sum_{t=1}^{m} \prod_{t'=1}^{n_t} k_{t,t'}$ where $k_{t,t'} \in \mathcal{K}_{\mathcal{B}}$. For simplicity, we use the notation γ to abstract both γ_p and γ_s in this section due to their symmetry. The conditional policy distribution $p(\gamma \mid \theta)$ is then parameterized using a standard Bayesian neural networks:

$$p(\gamma \mid \theta) \quad \sim \quad \mathcal{N}(\gamma; \mu(\theta), \Sigma(\theta)) \,, \tag{2.9}$$

where μ and Σ are neural networks that respectively generate the mean and covariance of the distribution.

As the generative trajectory encoded by θ is infinite, there is no analytical method to compute its optimal set of stopping points. However, we can approximate the optimal stopping point in this trajectory by finding the best performing intermediate kernel expression on the observed finite trajectory. Explicitly, let $\mathcal{S} = \{\{\bar{m}, \bar{n}_1, \bar{n}_2, \ldots, \bar{n}_m\} \mid \bar{m} \leq m, \forall t \in [\bar{m}] : \ \bar{n}_t \leq n_t\}$ be the set of all possible intermediate sets of stopping points that precede k, we define the *hindsight estimation* of k as:

$$\begin{aligned} k^* \quad &= \quad \sum_{t=1}^{m^*} \prod_{t'=1}^{n_t^*} k_{t,t'}, \quad \text{where} \\ \{m^*, n_1^*, \ldots, n_{m^*}^*\} \quad &\triangleq \quad \underset{\{m', n_1', \ldots, n_m'\} \in \mathcal{S}}{\arg\max} \; F_\tau \left(\sum_{t=1}^{m'} \prod_{t'=1}^{n_t'} k_{t,t'} \right) \,, \end{aligned} \tag{2.10}$$

and let K_τ^* denote the covariance matrix induced by k^* on training inputs the $\mathcal{D}$ described in τ. We argue that high-performing kernels likely produce covariance matrices that are similar to K_τ^* , which motivates the following loss function with respect to the current candidate weight θ :

$$\begin{aligned} \mathcal{L}_\theta(\mu, \sigma) \quad &= \quad \mathbb{E}_{\gamma \sim \mathcal{N}(\mu(\theta), \sigma(\theta))} \Big[\langle G(\theta, \pi(\gamma)), k^* \rangle_\tau \Big] \\ &\simeq \quad \frac{1}{r_\gamma} \sum_{i=1}^{r_\gamma} \langle G(\theta, \pi(\gamma_i)), k^* \rangle_\tau \,, \end{aligned} \tag{2.11}$$

where r_γ denotes the number of γ samples drawn from the conditional distribution; $\pi(\gamma_i)$ denotes the policy parameterized by γ_i drawn from the conditional policy distribution; $G(\theta, \pi(\gamma_i))$ denotes the kernel expression generated by G with weight θ and policy $\pi(\gamma_i)$); and $\langle k, k' \rangle_\tau \triangleq \|K_\tau - K'_\tau\|_{\text{Fro}}$ denotes the Frobenius norm of the difference between the covariance matrices induced by kernel functions k and k'.

This loss function, however, does not have an analytical gradient with respect to μ and σ as it requires simulation to compute. To optimize for μ and σ, we first employ the random gradient estimation technique [77], which approximates gradient at a point by evaluating the expected gradient of its υ-Gaussian smoothing. In particular, we derive our randomized gradient estimation for μ as follows:

$$\begin{aligned}
\nabla_\mu \mathcal{L}_\theta(\mu, \sigma) &\simeq \nabla_\mu \, \mathbb{E}_{\upsilon \sim \mathcal{N}(0, \mathbf{I})} \left[\ell_\theta(\mu + \upsilon, \sigma) \right] \\
&= \mathbb{E}_{\upsilon \sim \mathcal{N}(0, \mathbf{I})} \left[\ell_\theta(\mu + \upsilon, \sigma) \upsilon \right] \\
&\simeq \frac{1}{r_\upsilon} \sum_{j=1}^{r_\upsilon} \ell_\theta(\mu + \upsilon_j, \sigma) \upsilon_j \\
&\simeq \frac{1}{r_\gamma r_\upsilon} \sum_{i=1}^{r_\gamma} \sum_{j=1}^{r_\upsilon} \left\langle G\left(\theta, \pi(\mu_i + \upsilon_j, \sigma_i)\right), k^* \right\rangle_\tau \,, \qquad (2.12)
\end{aligned}$$

where r_υ denotes the number of υ samples drawn from the standard Gaussian distribution $\mathcal{N}(0, \mathbf{I})$ and we have rewritten $\pi(\gamma_i) = \pi(\mu_i, \sigma_i)$ to clearly show the perturbed component μ_i in the estimation. Similarly, the gradient estimation for σ is given as:

$$\nabla_\sigma \mathcal{L}_\theta(\mu, \sigma) \simeq \frac{1}{r_\gamma r_\upsilon} \sum_{i=1}^{r_\gamma} \sum_{j=1}^{r_\upsilon} \left\langle G(\theta, \pi(\mu_i, \sigma_i + \upsilon_j), k^* \right\rangle_\tau \,. \qquad (2.13)$$

These estimations allow us to update μ and σ via the gradient descent algorithm, which complete the specification of our policy update.

2.4 Empirical Study

This section evaluates and reports the empirical performance of our kernel selection framework DTERGENS on a synthetic kernel recovery task and kernel selection for regression on three

real-world datasets:

- The DIABETES dataset [25] contains 442 diabetes patient records (i.e., inputs) with 10 variables: age, sex, body mass index, average blood pressure and six blood serum measurements. The target output variable is a quantitative measure of disease progression one year after baseline.
- The MAUNA LOA dataset [52] measuring monthly average CO_2 concentration (in ppvm) of air samples at the Mauna Loa Observatory over 42 years (i.e., 504 observations in total).
- The PROTEIN dataset [82] with 45730 observations of protein structures, each records 9 physicochemical properties of a protein. The predictive variable is the size of the protein residue in kDa.

To demonstrate the performance of DTERGENS, we compare our method with the following benchmarks: (a) random search over the space of kernels with maximum length $L \leq 10$ (baseline); (b) SVO: Structure Variationally-Encoded Optimization [66], for which we train the VAE component using 25000 randomly generated kernel expressions with max length $L \leq 10$; and (c) DTERGENS with no stopping policy, for which we terminate the secondary component at random, and terminate the primary component upon reaching a maximum length $L = 2, 4, 6$. The baseline (b) serves to demonstrate the advantage of generative search, whereas (c) serves to demonstrate the advantage of having adaptive termination policies for the generative components. For all variants of DTERGENS, we additionally use the REMBO random projection trick [101] to address the high-dimensionality of the generative weights θ.

For all experiments, we demonstrate the performance of our framework on the black-box model Variational DTC Sparse Gaussian Process (vDTC) [36] with the following configurations: (1) 80/10/10 train-test-validation split (i.e., we use the validation fold to compute BO feedback and the test fold to evaluate final performance); (2) 100 randomly selected inducing inputs; (3) kernel hyper-parameters are optimized using L-BFGS over 100 iterations. These configurations implicitly define the learning scenario Ω, such that F_Ω is the root-mean-square-error (RMSE) of

predictions on the test split, given a model trained and validated accordingly on the train split. We construct the set of base kernels with 4 different base kernel functions, as suggested by Duvenaud et al. [23]. These kernel functions, along with their learnable parameters, are defined as follows:

$$k_{\text{LIN}}(\mathbf{x}_i, \mathbf{x}_j; \sigma_n, \sigma_b, \mathbf{c}) \triangleq \sigma_n^2 (\mathbf{x}_i - \mathbf{c})^\top (\mathbf{x}_j - \mathbf{c}) + \sigma_b^2 \tag{2.14}$$

$$k_{\text{SE}}(\mathbf{x}_i, \mathbf{x}_j; \sigma_n, \ell_1, \ell_2 \dots \ell_d) \triangleq \frac{1}{\sigma_n^2} \exp\left(\sum_{t=1}^{d} \frac{(\mathbf{x}_i^t - \mathbf{x}_j^t)^2}{\ell_t^2} \right) \tag{2.15}$$

$$k_{\text{PER}}(\mathbf{x}_i, \mathbf{x}_j; \sigma_n, \sigma_p, \ell_1, \ell_2 \dots \ell_d) \triangleq \frac{1}{\sigma_n^2} \exp\left(\sum_{t=1}^{d} \frac{2 \sin^2\left(\pi |\mathbf{x}_i^t - \mathbf{x}_j^t| / \sigma_p\right)}{\ell_t^2} \right) \tag{2.16}$$

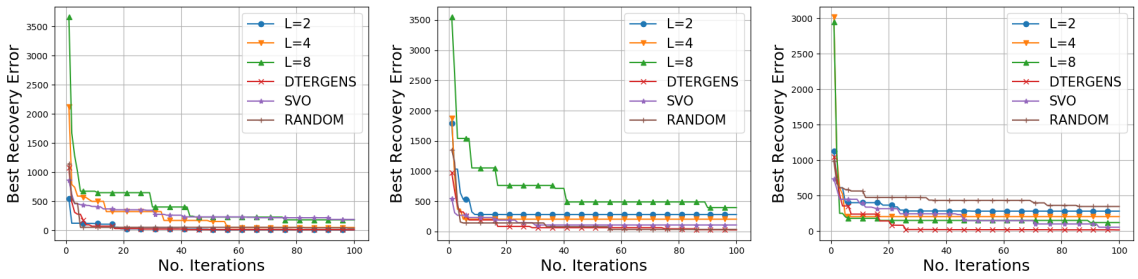
$$k_{\text{RQ}}(\mathbf{x}_i, \mathbf{x}_j; \sigma_n, \sigma_w, \mathbf{c}) \triangleq \sigma_n^2 \left(1 + \sum_{t=1}^{d} \frac{(\mathbf{x}_i^t - \mathbf{x}_j^t)^2}{2\sigma_w^2 \ell_t^2} \right)^{-\sigma_w} \tag{2.17}$$

We parameterize G_p and G_s using the same RNN architecture with 4 hidden feed-forward layers. Both G_p and G_s has hidden dimension $d_p = d_s = 5$. The emission output of G_p has dimension $d_s = 5$, as G_p is tasked to generate the initial hidden state of G_s, whereas the emission output of G_s has dimension $|\mathcal{K}_{\mathcal{B}}| = 4$, which corresponds to the number of base kernel functions. We use ReLU activation for all non-output layers, softmax activation for the kernel output layer of G_s and tanh activation for the emission output layer of G_p. Finally, we optimize our RNN parameters by adapting a known high-dimensional BO method called REMBO [101] to account for the dynamic function landscape (Section 2.3.3).

2.4.1 Synthetic Kernel Recovery

We first investigate how well various kernel selection methods recover a covariance matrix given synthetic data randomly drawn from its corresponding distribution. Unlike most real-world settings where a ground truth kernel is not known and performance evaluation relies on possibly noisy predictive accuracy, this scenario provides a ground truth for kernel selection and allows us to directly measure the success of various contending methods.

Explicitly, given an arbitrarily chosen kernel k^* (with arbitrarily initialized hyper-parameters) and n i.i.d. input observations $\tau = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \subset \mathbb{R}^d$ drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, we subsequently

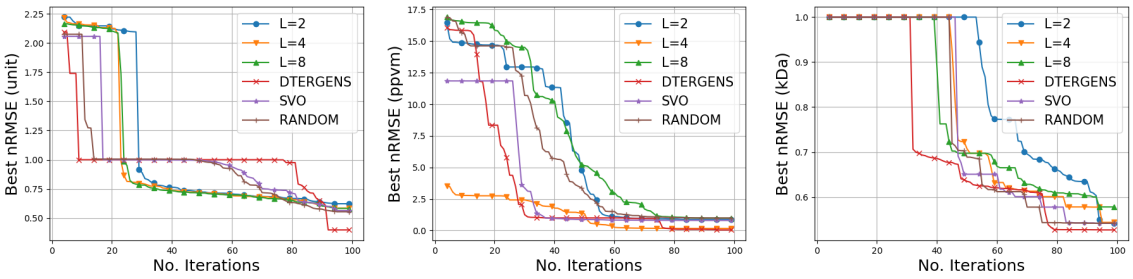


$\text{RQ} \times \text{RQ}$ $\text{PER} \times \text{RQ} \times \text{LIN} \times \text{LIN}$ $\text{LIN} \times \text{RQ} \times \text{LIN} + \text{PER} \times \text{LIN} + \text{RQ} \times \text{SE}$

Figure 2.3: Best kernel recovery error over 100 iterations with various kernel selection methods on three synthetic datasets constructed from specific kernels

generate corresponding output observations $Y = \{y_1, y_2 \dots y_n\}$, where $y_i \sim \mathcal{N}(0, K^*_\tau + \sigma^2 \mathbf{I})$ and K^*_τ denotes the data covariance matrix induced by k^*. We then apply various kernel selection methods, including DTERGENS for vDTC prediction on this synthetic dataset and measure our recovery error for any selected kernel k by $\mathcal{L}_{\text{rec}}(k) = \|K_\tau - K^*_\tau\|_{\text{Fro}}$. Fig. 2.3 shows the best recovery errors achieved over a span of 100 BO iterations with 3 different ground truth kernels: (1) $k^* = k_{\text{RQ}} \times k_{\text{RQ}}$; (2) $k^* = k_{\text{PER}} \times k_{\text{RQ}} \times k_{\text{LIN}} \times k_{\text{LIN}}$; and (3) $k^* = k_{\text{LIN}} \times k_{\text{RQ}} \times k_{\text{LIN}} + k_{\text{PER}} \times k_{\text{LIN}} + k_{\text{RQ}} \times k_{\text{SE}}$.

In all experiments, DTERGENS consistently achieves the lowest recovery error after 100 iterations compared to other methods. Random search performs competitively when the ground truth kernels are simple (i.e., $L = 2, 4$), and hence are easy to be found via randomization. On the other hand, random search expectedly performs the worst when the ground truth kernel is longer (i.e., $L = 7$). We also observe that without the termination policy component, the performance of DTERGENS is only competitive when L is set to be roughly the length of the ground truth kernel, but otherwise outperformed by other methods. This shows the importance of adaptively learning the complexity of the kernel expression using a data driven policy. Lastly, we observe that SVO is most significantly outperformed by DTERGENS in the first experiment. We reason that this is because the number of length-2 kernels is relatively smaller in the set of training expressions for



DIABETES MAUNA LOA PROTEIN

Figure 2.4: Best nRMSE over 100 iterations with various kernel selection methods on 3 benchmark datasets using vDTC [36].

the VAE component of SVO. Thus, the trained VAE could be biased to produce longer kernels and it is more difficult for SVO to find a latent embedding that decodes to a length-2 kernel. In contrast, DTERGENS does not incur this problem because its termination policy is also learned as it collects information about the embedding space.

2.4.2 Kernel Selection for Regression Tasks

This section investigates the performance of kernel selection for regression tasks using vDTC [36] on DIABETES [25], MAUNA [52] and PROTEIN [82] datasets. In all experiments, we measure performance by computing the root-mean-square-error of predictions, normalized against the root-mean-square-error achieved by fixing the kernel of VDTC to be k_{SE}, which serves to demonstrate the improvement over the default choice of kernel. Explicitly, our kernel selection metric for any selected kernel k is given by:

$$\text{nRMSE}(k) \triangleq \sqrt{\frac{\sum_{i=1}^{n_{\text{test}}} \left(\bar{y}_i(k) - y_i\right)^2}{\sum_{i=1}^{n_{\text{test}}} \left(\bar{y}_i(k_{SE}) - y_i\right)^2}} \tag{2.18}$$

where $\bar{y}_i(k)$ denotes the prediction made by vDTC for test input $\mathbf{x}_i$ with selected kernel function k and y_i denotes the corresponding ground truth test output.

Fig. 2.4 shows the comparative performance between DTERGENS and the competing methods. Across all datasets, DTERGENS consistently obtains the best performing kernel expression.

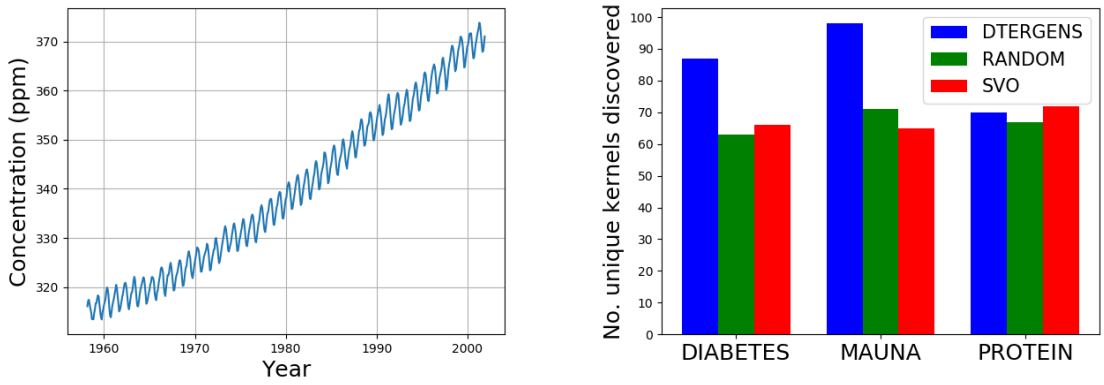


Figure 2.5: (Left) the linear-periodic trend of the MAUNA dataset; and (Right) the number of unique kernels discovered by DTERGENS, SVO and random search on all three datasets.

On the PROTEIN dataset, DTERGENS also shows the fastest convergence among all competing methods. On the MAUNA dataset, DTERGENS performs competitively with $L = 4$ and both variants of DTERGENS outperform SVO. More interestingly, the best kernel found for the MAUNA dataset is $k_{\text{LIN}} \times k_{\text{PER}} \times k_{\text{PER}} + k_{\text{RQ}} \times k_{\text{PER}}$, which accurately reflects the linearly increasing periodic nature of the data (Fig. 2.5 (left)).

Fig. 2.5 (right) further compares the expressiveness of the three kernel selection methods (i.e., DTERGENS, SVO and random search), which is measured by the number of unique kernels found over 100 iterations in each method. As expected, random search consistently produces the same amount of unique expressions across all experiments. While SVO discovers approximately the same amount of unique kernels as does random search on all three datasets, it tends to outperform random search as its discovery is guided. Finally, we observe that DTERGENS consistently discovers more unique kernels and also outperforms the other methods. This finding asserts our earlier intuition on how adding expressiveness to the embedding method also helps to improve search efficiency.

2.5 Conclusion

We tackle the composite kernel selection problem for an arbitrary kernel-based model. The proposed DTERGENS algorithm reformulates the kernel search problem as a parameter optimization task for a recursive kernel generator equipped with optimal stopping mechanism. Operating on this well-behaved space of generative weights allows efficient traversing of the candidate space with Bayesian Optimization. We demonstrate that DTERGENS discovers complex kernels that improves the model performance. The performance gain is significantly more than that of other state-of-the-art kernel learning methods on various real-world datasets.

Orthogonal to this work, we have also studied another aspect of model configuration for the GP framework to improve the accuracy-runtime trade-off. In particular, the classical GP algorithm [83] has a $\mathcal{O}(n^3)$ run-time complexity, where n is the size of the training data. Due to this prohibitively expensive cost, many sparse approximations have been employed to improve the scalability of GP in practical use cases. These methods, which are broadly referred to as sparse Gaussian processes (SGPs), usually focus on optimizing a compact set of inducing data points that serve as sufficient statistics for the training data [33, 36, 42]. Nonetheless, previous theoretical results have established that this inducing set should be sufficiently large for the SGP predictions to closely approximate that of GP, and it therefore presents an accuracy-efficiency trade-off inherent to most SGP variants.

We study this trade-off and derive a set of conditions for the training data of the sparse spectrum GP (SSGP) method [33]. When these conditions are satisfied, the SSGP method only requires a compact set of inducing inputs to closely approximate the full GP prediction. Based on this theoretical understanding, we further develop a practical approach to recondition the training data using a variational autoencoder (VAE) approach [53]. We showed that our VAE-transformed data exhibit the proposed conditions and achieve better sample complexity than vanilla SSGP. This work is co-authored by Nghia Hoang, Hai Pham and David Woodruff, and was published at the Conference on Neural Information Processing Systems (NeurIPS) 2020 [40].

Chapter 3

Minimizer Sketch Design

3.1 Introduction

The minimizer scheme [86, 89] is a deterministic method that samples length-k substrings (i.e., k-mers) from a long sequence such that sufficient information about the identity of the sequence is preserved for alignment purposes. Given a choice of k and a window size w, the minimizer scheme selects the lowest priority k-mer from each overlapping window in the target sequence according to some total ordering π over all k-mers. The set of all k-mers selected in this manner is called a minimizer sketch. Minimizer sketches are widely used to reduce memory consumption and run-time in bioinformatics applications such as genome assemblers [106], read mappers [48, 59] and k-mer counters [20, 27].

The performance of a minimizer sketch can be measured by its density [70] (i.e., the sketch size relative to the target sequence length, lower is better), or its conservation [24] (i.e., the expected fraction of k-mers that remain in the sketch over random homologous copies of the target sequence, higher is better) with respect to the target sequence. Depending on the choice of π, the resulting density and conservation can vary significantly. The theoretical lower-bound of density achievable by any minimizer scheme is given by $\mathcal{O}(1/w)$ [70]. Given a target sequence with uniformly random characters, a uniformly random ordering π will yield an expected density of

$\mathcal{O}(2/w)$ [89]. On the other hand, no concrete guarantees have been devised for the conservation metric of minimizer sketches. To find k-mer sketches with better conservation, Edgar [24] proposes the syncmer scheme, which was subsequently proven to have better expected conservation than minimizers on comparable settings [90].

For applications that involve aligning many queries against a single reference, such as genome assembly, it is often more desirable to optimally design an ordering that performs well on the reference sequence. The idea of learning minimizer schemes for a specific target sequence was originally explored in the context of the density metric and manifested as a challenging permutation optimization objective [41]. Existing approaches approximate this task by discrete surrogate objectives that are either inaccurate [15, 18, 47], or not necessarily easier to solve [111]. There are no prior works that have successfully tackled the optimization of the sequence-specific conservation metric.

To address these problems, we introduce a novel approximation of the density minimizing objective. Specifically, we reformulate the original permutation learning problem as parameter optimization of a deep learning system called DEEPMINIMIZER. This results in the first differentiable objective for minimizer sketch design that can be efficiently optimized using gradient-based learning techniques. We subsequently extend this method to the MASKEDMINIMIZER algorithm that accounts for the joint optimization of density and conservation for minimizers as well as its various generalizations.

The first part of this chapter details the DEEPMINIMIZER algorithm, which implicitly represents k-mer orderings using a continuous scoring function that is parameterized by a neural network called PRIORITYNET. The PRIORITYNET assigns a consistent score to each k-mer regardless of its position in the sequence, and hence guarantees the recovery of a total k-mer ordering given the assigned scores. This property ensures that the set of selected k-mers corresponds to a valid minimizer sketch. As the density objective is discrete and not amenable to gradient-based learning, we further design another neural network called TEMPLATENET.

The TEMPLATENET potentially outputs *inconsistent* k-mer scoring, but otherwise guarantees that few k-mers will be selected. This property is complementary to that of the PRIORITYNET, which enforces consistent scoring, yet does not necessarily yield a low density sketch. The DEEPMINIMIZER algorithm thus approximates the original density objective via minimizing a locality-aware distance metric between the outputs of these networks, resulting in a consensus sketch that is both consistent and sparse.

The second part of this chapter details the MASKEDMINIMIZER algorithm, which extends the DEEPMINIMIZER framework to account for a generalized formulation of the minimizer scheme, as well as the capacity to optimize for the conservation metric. Inspired by Dutta et al. [22], the MASKEDMINIMIZER method additionally defines a mask variable that encodes a set of positions, such that the lowest ordered k-mer in each window is only sampled if its relative position is found in the mask. This pattern-aware sampling rule enriches minimizers with the ability to (1) balance between density and conservation; and (2) to avoid dense sketches of adversarially patterned subsequences. Both properties are achieved by optimally configuring the mask variable. Last, we introduce a bi-level optimization algorithm that alternates between pruning the mask variable and learning the k-mer ordering. Given a fixed mask, the inner loop optimizes for the k-mer ordering via an extension of the DEEPMINIMIZER objective function. On the other hand, the outer loop searches for the optimal mask via greedily pruning its set bits, forwarding pruned candidates to the inner loop, and selecting one that yields the best performance gain.

We demonstrate the performance of both methods on a collection of benchmark genomic sequences, including several human and bacterial genomes. The DEEPMINIMIZER method yields sketches with significantly better density than those obtained by other existing approaches. To better capture the multi-objective optimization performance of the MASKEDMINIMIZER method, we further introduce a new evaluation metric, called generalized sketch score (GSS), that combines and measures the trade-off between density and conservation. We show that the optimized masked minimizer sketches achieve better GSS than previous optimization ap-

proaches, including the DEEPMINIMIZER method. In addition, we also discover a specific class of complement mask patterns that exhibit desirable properties in sketching homopolymer-rich sequences.

The original idea of the DEEPMINIMIZER method was co-authored by Carl Kingsford and Hongyu Zheng, and was published at the Conference on Research in Computational Biology (RECOMB) in 2021 [41]. It was subsequently extended and published at the Journal of Computational Biology (JCB) in 2022. The MASKEDMINIMIZER method was co-authored by Carl Kingsford and Guillaume Marçais, and was accepted to the Journal of Computational Biology (JCB) in 2022.

3.2 Problem setting

3.2.1 Notation

Let Σ be an arbitrary alphabet over which an input sequence $S \in \Sigma^L$ is defined. We further let κ_i^k and $L_k \triangleq L - k + 1$ respectively denote the i^{th} k-mer and the total number of overlapping k-mers in S. A (w, k)-window is a substring of length $w_k \triangleq w + k - 1$ and contains exactly w overlapping k-mers. By extension, the i^{th} (w, k)-window and the total number of (w, k)-windows are denoted by $\kappa_i^{w_k}$ and $L_{w_k} = L - w_k + 1$. Finally, let π be some k-mer ordering such that $\kappa_i^k \prec_\pi \kappa_j^k$ implies κ_i^k precedes κ_j^k in π, we additionally define an index selector function $m_\pi(a, w) \triangleq \underset{i \in [0, w-1]}{\operatorname{argmax}} \sum_{j=0}^{w-1} \mathbb{I}(\kappa_{a+i}^k \prec_\pi \kappa_{a+j}^k)$ that returns the lowest-ranked k-mer in the w-window starting at κ_a^k.

Definition 4 (Minimizers) *The k-mer sampling minimizer scheme is characterized by a tuple of parameters (w, k, π). π denotes a total ordering over the set of all k-mers. The minimizer method samples and reports the indices of the lowest-ranked k-mers (e.g., minimizers) from each (w, k)-window in S:*

$$\mathcal{M}(S; w, k, \pi) \triangleq \{i + m_\pi(i, w)\}_{i \in [1, L_{w_k}]}. \tag{3.1}$$

Evaluation metrics. Minimizer sketches are evaluated using various metrics. Schleimer et al. [89] uses the *density* metric to measure the compression factor which approximates the degree of cost-saving in downstream applications. More recently, Edgar [24] proposes the *conservation* metric (i.e., the likelihood of sketched k-mers to be persistently sampled across homologous sequences) and argues that high conservation is preferable when comparing sequences that might have diverged due to mutations and/or sequencing error. This thesis further introduces the *coverage* metric that measures the spread of the sketch across the input sequence. We note that the minimizer sketch has maximum coverage by design, but other sequence sketching schemes such as open syncmers [24], parameterized syncmers [22] or the generalized masked minimizers introduced later in this chapter may not.

Definition 5 (Density) *Let $\mathcal{X}$ be an arbitrary k-mer sampling scheme parameterized by θ. The density metric [70] measures the size of the sketch $\mathcal{X}(S;\theta)$ relative to the number of k-mer in S (lower is better):*

$$D(S;\mathcal{X},\theta) \triangleq \frac{1}{L_k}|\mathcal{X}(S;\theta)| \,. \tag{3.2}$$

Definition 6 (Conservation) *Let S' be a homologous sequence to S (e.g., differing by a few random base substitutions), and suppose S' follows some arbitrary distribution p_S. The conservation metric [24] measures the expected number of bases that are present in both $\mathcal{X}(S;\theta)$ and $\mathcal{X}(S';\theta)$, relative to the number of k-mers in S (higher is better). For ease of comparison to the density metric, we instead define the conservation metric in terms of the number of sketched k-mers:*

$$C(S;\mathcal{X},\theta) \triangleq \frac{1}{L_k}\mathbb{E}_{S'\sim p_S}\left|\mathcal{X}(S;\theta)\cap\mathcal{X}(S';\theta)\right| \,. \tag{3.3}$$

Definition 7 (Coverage) *The w-coverage metric computes the fraction of (w, k)-windows that overlap at least one sampled k-mer in $\mathcal{X}(S; \theta)$. This means a minimizer sketch is guaranteed to have a w-coverage value of 1 by construction; whereas an empty sketch has a w-coverage value of 0. The w-coverage metric is formally given by:*

$$\begin{aligned} V_w(S; \mathcal{X}, \theta) &= \frac{1}{L_{w_k}} \sum_{i=1} V_w^i(S; \mathcal{X}, \theta) \\ &\triangleq \frac{1}{L_{w_k}} \sum_{i=1}^{L_{w_k}} \left(1 - \prod_{j=i}^{i+w-1} \mathbb{I}(j \notin \mathcal{X}(S)) \right), \end{aligned} \tag{3.4}$$

where V_w^i indicates the event $\kappa_i^{w_k}$ overlaps at least one sampled location in $\mathcal{X}(S)$.

The Minimizer Sketch Design (MSD) problem can be broadly stated as finding an ordering π, or equivalently a scoring function f_π, that optimizes some of the above metrics for a given target sequence. In this thesis, we consider two variants of the Minimizer Sketch Design (MSD) problem. The first variant corresponds to minimizing the density metric. The second variant extends the search space to minimizers whose sampling patterns are more generally encoded by a binary mask parameter, and optimizes for a multi-objective metric combining density, conservation and coverage. We defer the technical definition of each variant to its corresponding section.

3.2.2 Other sketching variants

Edgar [24] proposes the open-syncmer scheme to address the goal of shifting the focus from minimizing density to maximizing conservation, which is preferable when comparing sequences that might have diverged due to mutations and/or sequencing error. Shaw and Yu [90] subsequently shows that an open syncmer scheme has provably better expected conservation than a minimizer scheme when both their orderings are uniformly random. However, there are currently no methods to optimize the open-syncmer ordering with respect to either density or conservation.

Definition 8 (Open syncmers) *The k-mer sampling open-syncmer scheme [24] is specified by a tuple (k, s, t, π). Here, the parameter $s < k$ implicitly characterizes the representation of k-mers*

as the collection of their constituent s-mers. We additionally denote the number of s-mers in each k-mer by $k_s \triangleq k - s + 1$. The parameter π denotes a total ordering over the set of s-mers. Finally, the offset parameter $0 \leq t \leq k - 1$ indicates that the scheme will sample all k-mers in which the lowest-ranked constituent s-mer is exactly at position t (relative to the k-mer position):

$$\mathcal{O}(S; k, s, t, \pi) \triangleq \{i \mid m_\pi(i, s) = t\}_{i \in [1, L_k]} . \tag{3.5}$$

Based on the syncmer concept, Dutta et al. [22] subsequently introduces the parameterized syncmer scheme, which replaces t by a subset of qualifying offset positions $v \subseteq [0, k_s - 1]$. Setting $v = \{t\}$ for some $t \in [k_s - 1]$ recovers the open-syncmer scheme above. This flexible encoding of sampling rules offers a practical handle on the performance of syncmers, where subsets that do not correspond to open-syncmers have been shown to achieve superior performance [22].

Definition 9 (Parameterized syncmers) *The parameterized syncmer method samples and reports the indices of all k-mers such that their lowest-scoring s-mers are found at some offset positions in v:*

$$\mathcal{O}^+(S; k, s, v, \pi) \triangleq \{i \mid m_\pi(i, k_s) \in v\}_{i \in [1, L_k]} . \tag{3.6}$$

3.2.3 Other MSD approaches

Universal hitting set construction. Most existing minimizer selection schemes with performance guarantees over random sequences are based on the theory of universal hitting sets (UHS) [71, 79]. Particularly, a (w, k)-UHS is defined as a set of k-mers such that every window of length w (from any possible sequence) contains at least one of its elements. Every UHS subsequently defines a family of corresponding minimizer schemes whose expected densities on random sequences can be upper-bounded in terms of the UHS size [70]. As such, to obtain minimizers with

provably low density, it suffices to construct small UHS, which is the common objective of many existing approaches [26, 70, 110]. These methods, however, rely on the unrealistic assumption that the target sequences follow a uniform distribution [109]. As such, there tends to be little correspondence between the provable upper-bound on expected density and the actual density measured on a target sequence.

Heuristic methods. Several minimizer construction schemes rank k-mers based on their frequencies in the target sequence [15, 47], such that infrequent k-mers are more likely to be chosen as minimizers. These constructions nonetheless rely on the assumption that infrequent k-mers are spread apart and ideally correspond to a sparse sampling. Another greedy approach is to sequentially remove k-mers from an arbitrarily constructed UHS, as long as the resulting set still hits every w-long window on the target sequence [18]. Though this helps to fine-tune a given UHS with respect to the sequence of interest, there is no guarantee that such an initial set will yield the optimal solution after pruning.

Polar set construction. Recently, a novel class of minimizer constructions was proposed based on polar sets of k-mers, whose elements are sufficiently far apart on the target sequence [111]. The sketch size induced by such a polar set is shown to be tightly bounded with respect to its cardinality. This reveals an alternate route to low-density minimizer schemes through searching for the minimal polar set. Unfortunately, this proxy objective is NP-hard and currently approximated by a greedy construction [111], which can be sub-optimal in practice.

3.3 Density optimization of the minimizer scheme

This section presents an approach to solve the problem of low-density minimizer sketch selection. In general, a low-density minimizer scheme achieves three desiderata: (1) the sketch is compact and will offer significant cost saving to downstream applications; (2) every (w, k)-window in S overlaps at least one k-mer in the sketch; and (3) identical windows are represented by the same k-mer due to the deterministic sampling protocol. These properties give rise to the following

problem definition.

Definition 10 (Low density MSD) *Let $S \in \Sigma^L$ be a length-L sequence drawn from the vocabulary Σ. Assuming that w and k are fixed parameters of the minimizer scheme, the density minimizing MSD objective can be written as an optimization task with respect to the k-mer ordering π:*

$$\pi_* \quad = \quad \underset{\pi \in \Pi(\Sigma^k)}{\operatorname{argmin}} \, D(S; \mathcal{M}, \pi) \, , \tag{3.7}$$

where $\Pi(\Sigma^k)$ denotes the set of all k-mer orderings.

Generally, this task is challenging due to a factorial large search space (e.g., consisting of $|\Pi(\Sigma^k)| = |\Sigma|^k!$ possible orderings), and the discrete nature of the density minimizing objective in Eq. (3.7). To work around these issues, existing methods typically surrogate the pairwise precedence operator $\prec_\pi$ with a more easily optimized scoring heuristic f such that $\mathbb{I}(f(\kappa_i^k) < f(\kappa_j^k))$ approximates $\mathbb{I}(\kappa_i^k \prec_\pi \kappa_j^k)$.

Many heuristics for f have been previously proposed. For example, Chikhi et al. [15] and Jain et al. [47] construct f from relative k-mer frequency, that is $f(\kappa_i^k) \propto \sum_{j=1}^{L_k} \mathbb{I}(\kappa_j^k = \kappa_i^k)$. Ekim et al. [26] and Zheng et al. [110] employ a universal hitting set (UHS) υ to characterize f, such that for any $\kappa_i^k \in \upsilon$ and $\kappa_j^k \notin \upsilon$, it is guaranteed that $f(\kappa_i^k) < f(\kappa_j^k)$. Similar set-ups have been explored in the context of sequence-specific minimizers using a pruned UHS $\upsilon(S)$ [18] and a polar set $\zeta(S)$ [111] constructed for the target sequence. Here, we note that the notation f is overloaded to admit different parameter representations. This is mainly to highlight the unification of existing methods, and has no implication on the mathematical consistency of our formulation.

These methods can be seen as crude approximations of π that map k-mers to a small number of discrete buckets (e.g., fewer than $|\Sigma|^k$). When collision of values occur (e.g., different k-mers are assigned to the same bucket), these algorithms rely on a pre-determined ordering (e.g., typically random) to break ties, and thus once again may not yield optimal density. To achieve a more fine-grained representation of π, we instead model f using a neural network parameterized

by weights α. That is, f takes as input the multi-hot encoding of a k-mer, and returns a corresponding real-valued score in $[0, 1]$. As continuous scores are less likely to collide, this scheme practically eliminates collisions and allows the recovery of a total ordering from the score assignment. Using the notion of this neural function f, we can succinctly rewrite the selector function in Section 3.2.1 as:

$$m_f(a, w) \triangleq \underset{i \in [0, w-1]}{\operatorname{argmin}} f(\kappa^k_{a+i}; \alpha) \,, \tag{3.8}$$

which reparameterizes the objective in Eq. (3.7) as an optimization task with respect to weight α.

Practically, sequentially applying this network on every k-mer in S can be efficiently written as a single convolutional neural network (CNN). To differentiate this from the atomic function f, we denote the output of the CNN as $\mathbf{f}(S; \alpha) \triangleq [f(\kappa^k_i; \alpha)]_{i \in [L]}$. We require that the score assignment induced by the CNN f to be *consistent* across different windows in order to recover a valid ordering π. Specifically, one k-mer can not be assigned different scores at different locations in S. To enforce this, we let the first convolution layer of our architecture, PRIORITYNET, have kernel size k, and all subsequent layers to have kernel size 1. This design ensures that the output entry corresponding to a k-mer is only dependent on the encoding of that k-mer alone. An illustration for $k = 2$ is given in Fig. 3.1.

3.3.1 Proxy Objective

The reparameterized objective, however, remains non-differentiable with respect to the network weights α. As such, α cannot be readily optimized with gradient back-propagation techniques typically used in most deep learning frameworks. To work around this, we introduce a proxy optimization objective that couples the PRIORITYNET with another function called the TEMPLATENET, which relaxes the *consistency* requirement and generates *template* score assignments that do not necessarily recover a valid minimizer schemes. In exchange, such *templates* are guaranteed to yield low densities by design.

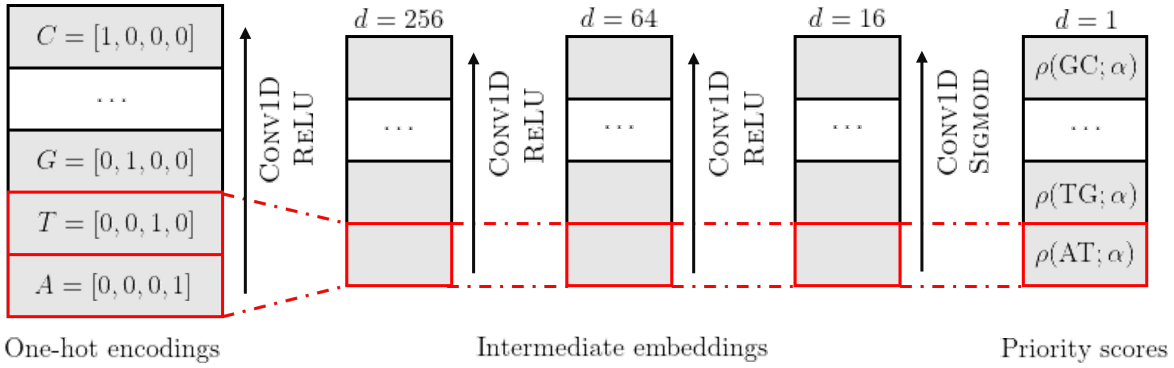


Figure 3.1: Our PRIORITYNET architecture for $k = 2$, parameterized by weights α, maps sequence multi-hot encoding to priority scores through a series of 3 convolution layers with kernel size $[k, 1, 1]$ and $[256, 64, 16]$ embedding channels respectively. Given a fixed network weights α, the priority score of any k-mer is therefore deterministic and does depend on its position in the sequence.

Intuitively, the goals of these networks are complementary: the PRIORITYNET induces a valid minimizer scheme via a *consistent* priority score assignment, whereas the TEMPLATENET pinpoints potential neighborhoods of *low-density* score assignments. This reveals an alternative optimization pathway in which this pair of networks negotiate towards a consensus solution that (a) satisfies the consistency constraint enforced by PRIORITYNET; and (b) resembles a template in the output space of TEMPLATENET, and thus will potentially yield low density.

Specifically, we let the TEMPLATENET be characterized by a parameterized function g that maps any k-mer index in $[L]$ to a real-valued score in $[0, 1]$. The output of the TEMPLATENET is achieved by applying g on every k-mer, and is denoted by $\mathbf{g}(S; \beta) = [g(i; \beta)]_{i \in [L]}$, where β is the weights of g. Our objective can be cast as minimizing a distance metric Δ between $\mathbf{f}$ and $\mathbf{g}$:

$$(\alpha_*, \beta_*) \quad = \quad \underset{\alpha,\beta}{\operatorname{argmin}} \, \Delta \left(\mathbf{f}(S; \alpha), \mathbf{g}(S; \beta) \right) \ . \tag{3.9}$$

We subsequently detail the full specification of our proxy objective, which includes the parameterization of the TEMPLATENET (Section 3.3.2), which ensures that any output template approximates the theoretical lower-bound density [70] on the target sequence, and the practical

choices of the distance metric Δ to accurately capture the negotiation objective between the two networks.

3.3.2 Specification of TemplateNet

The well-known theoretical lower bound $1/w$ for density implies that the optimal minimizer, if it exists, samples k-mers exactly w positions [70]. As a result, we will construct g such that $\mathbf{g}(S;\beta)$ approximates this uniform assignment pattern given any initialization of its parameter β. Proposition 1 below shows a sufficient construction of g such that $\mathbf{g}(S;\beta)$ approximately yields the optimal density.

Proposition 1 *Let $g : \mathbb{R} \to [0,1]$ be a periodic function, with fundamental period w, such that g has a unique minimum value on every w-long interval. Formally, h satisfies:*

$$\forall t \in \mathbb{R} \quad : \quad g(t) = g(t+w) \tag{3.10}$$

$$\forall i,j \in \operatorname*{arginf}_t h(t),\ \exists u \in \mathbb{N} \quad : \quad |i-j| = uw\ . \tag{3.11}$$

Then, the template $\mathbf{g}(S;\beta)$ induces a sketch with density factor $1/w + o(1)$ on S when S is sufficiently long (i.e., $L_w \gg w^2$).

Proof: Even though g may not satisfy the consistency constraint, and hence is not a valid minimizer scheme, it still induces a k-mer sampling scheme that we denote by $\mathcal{G}$. Further let $\gamma_1 \triangleq 1$ and γ_t indicates the event that $\mathcal{G}$ picks a different k-mer in the t^{th} window than in the $(t-1)^{\text{th}}$ window. That is, $\gamma_t \triangleq \mathbb{I}(m_g(t,w) \neq m_g(t-1,w))$. Then, the density of $\mathcal{G}$ can be expressed as:

$$D(S;\mathcal{G},\beta) \quad = \quad \frac{1}{L_w}\sum_{t=1}^{L_w}\gamma_t\ . \tag{3.12}$$

For any value of $u \in \mathbb{N}^+$, we further define the integer interval $\mathcal{I}_u \triangleq [(u-1)w+1, uw]$. As the density of the entire sequence is simply the sum of density for each interval $\mathcal{I}_u$, it is then sufficient to derive the values of γ_t for all values of t in some arbitrary interval $\mathcal{I}_u$.

Without loss of generality, we assume $0 \in \operatorname*{arginf}_t g(t)$ since this can always be achieved via adding a constant phase shift to g. As g has a period of w, this implies $\{uw \mid u \in \mathbb{N}^+\} \subseteq \operatorname*{arginf}_t g(t)$, which further reduces to $\{uw \mid u \in \mathbb{N}^+\} \equiv \operatorname*{arginf}_t g(t)$ when condition (2) holds. Then, it follows that $\forall t \neq uw$, we have $t \notin \operatorname*{arginf}_t g(t)$. In addition, the index uw is in the window $\kappa_t^{w_k}$ by definition. Together, these facts imply that $\forall t : m(t, w) = uw$, and consequently $\gamma_t = 0$ for all $t \neq (u-1)w + 1$ since the index uw is overlapped by $\kappa_{(u-1)w+1}^{w_k}$.

In addition, for $u = 1$, we trivially have $\gamma_{(u-1)w+1} = \gamma_1 = 1$ by definition. For any $u > 1$, we have $m(\kappa_{(u-1)w}^{w_k}) = (u-1)w$, and $m(\kappa_{(u-1)w+1}^{w_k}) = uw$, which alo implies that $\gamma_{(u-1)w+1} = 1$. Finally, using the above derivations, we have:

$$D(S; g) = \frac{1}{L_w} \left(c + \sum_{u=1}^{\lfloor \frac{L_w}{w} \rfloor} \sum_{t \in \mathcal{I}_u} \gamma_t \right) = \frac{1}{L_w} \left(c + \left\lfloor \frac{L_w}{w} \right\rfloor \right) , \tag{3.13}$$

where $c \triangleq \sum_{t=\lfloor \frac{L_w}{w} \rfloor w+1}^{L_w} \gamma_t$ is the remainder of the sequence that does not make up any complete interval. The second equality follows from the derived values of γ_t for $t \in \mathcal{I}_u$. Finally, using the fact that $c = L_w - \lfloor \frac{L_w}{w} \rfloor w < w$ and the sufficient length assumption $L_w \gg w^2$, we have:

$$\frac{1}{L_w} \left(c + \left\lfloor \frac{L_w}{w} \right\rfloor \right) \;<\; \frac{1}{w} + \frac{w}{L_w} \;=\; \frac{1}{w} + o(1) , \tag{3.14}$$

which concludes our proof. □

While this sketch has guaranteed low density, it does not preserve the sequence identity like a minimizer sketch, hence is not useful for downstream applications. However, it is sufficient as a guiding template to help PRIORITYNET navigating the space of orderings. By Proposition 1, TEMPLATENET can be as simple as $g(t) = \sin(2\pi t/w)$ to induce a near-optimal score assignment. This naïve specification, however, encodes exactly a single set of template minima (i.e., one that picks k-mers from the set of interval positions $\{w, 2w, \dots\}$), which might not be in proximity of any valid minimizer scheme. For example, consider a sequence S in which some particular k-mer uniquely occurs at positions $t \in \left\{\frac{1}{2}w, \frac{3}{2}w, \dots\right\}$. The ideal assignment would be such that minima will occur at these locations, which is impossible.

It is therefore necessary that the specification of TEMPLATENET is sufficiently expressive for Eq. 3.9 to find an optimal solution. To model this family of template functions, we subsequently propose several parameterization strategies using (1) an ensemble of sinusoidal functions with integer phase shifts or (2) a Fourier series model that encodes any arbitrary sinusoidal function. We further propose an independent positional phase-delay component that can be combined with (1) and (2) to encode template functions with approximately constant period.

Ensemble Template Model

We first give a construction of a periodic model such that every k-mer position appears in at least one template encoded by its parameter space. To achieve this, we employ a linear combination of multiple sine functions with fixed integer phase shifts $\phi \in [w-1]$, each of which encodes a set of minima with a unique positional offset such as $\mathcal{T}_1 = \{0, w, 2w, \dots\}, \mathcal{T}_2 = \{1, w+1, 2w+1, \dots\}, \dots\ \mathcal{T}_{w-1} = \{w-1, 2w-1, 3w-1, \dots\}$. In particular, we define:

$$g(t;\beta) \triangleq \sigma\left(\sum_{\phi=0}^{w-1} \beta_\phi \sin\left(\frac{2\pi}{w}(t+\phi)\right)\right), \tag{3.15}$$

where the sigmoid activation function σ ensures that $h(t)$ appropriately maps to $[0,1]$ and outputs scores on the same scale as PRIORITYNET; $\beta = \{\beta_\phi\}_{\phi=0}^{w-1}$ are optimizable amplitude parameters such that $\beta_\phi \geq 0$ and $\sum_{\phi=1}^{w} \beta_\phi = 1$. Optimizing β then determines the dominant phase shift $\phi_{\max} = \operatorname{argmax}_\phi \beta_\phi$, which in turn controls the final offset of the template minima. Additionally, allowing the amplitudes of the ensemble components to be optimizable also helps to generate sufficient slack room for matching the template scores against the priority scores.

Truncated Fourier Series Template Model

The periodic function $g(t)$ with period w can be generalized using a Fourier series, which is a linear combination of an infinite number of sine and cosine functions, whose frequencies are

integer multiples of $1/w$:

$$g(t;\beta) \;=\; \sigma\left(\beta_0 + \sum_{r=1}^{\infty}\left[\beta_{r,1}\sin\left(\frac{2r\pi}{w}t\right) + \beta_{r,2}\cos\left(\frac{2r\pi}{w}t\right)\right]\right)\;, \tag{3.16}$$

where $\beta = \{\beta_{r,1}, \beta_{r,2}\}_{r=0}^{\infty} \cup \{\beta_0\}$ are optimizable amplitude parameters. For computational efficiency, we approximate g by a finite truncation up to the first R summands of the above Fourier series:

$$g(t;\beta) \;\simeq\; \sigma\left(\beta_0 + \sum_{r=1}^{R}\left[\beta_{r,1}\sin\left(\frac{2r\pi}{w}t\right) + \beta_{r,2}\cos\left(\frac{2r\pi}{w}t\right)\right]\right)\;. \tag{3.17}$$

Similar to the ensemble template model, optimizing the amplitude parameters β of this model also determines the offset of the minima locations and adds slack room to help matching against the priority score assignment. The key difference between these two template models is that the ensemble model requires all w phase shifts (and hence, all w component functions) to encode every k-mer location, whereas the Fourier model can achieve the same with a fixed value of R and remains compact even for large w. The Fourier model, however, will admit periodic functions whose minima do not coincide with integer indices, therefore condition (2) above will be less likely to hold in practice.

Positional Phase-Shift Model

By Proposition 1, all template score assignments encoded by the above β-parameterized families of functions correspond to near-optimal minimizer schemes with approximately perfect density factors. However, we note that this set of template solutions is usually unrealistic and cannot be mirrored exactly by PRIORITYNET, especially on complex problem instances with more difficult scoring constraints. For example, while the theoretical lower bound for densityis $1/w$, the actual optimal density factor attainable given a specific sequence is often considerably larger and occurs when consecutive minimizer locations are not always exactly w locations apart.

Motivated by this observation, we further extend our template model with a learnable component that adaptively adjusts the local frequencies of every encoded periodic function through

adding positional noise to their phase shift parameters. That is, let $\xi(S;\gamma) \in [-1,1]^L$ be a noise generating function parameterized by γ and let $\xi_i(S;\gamma)$ be the noise value corresponding to the i^{th} k-mer. We define the (ϵ,γ)-augmented TEMPLATENET as:

$$\mathbf{g}(S;\beta,\gamma) \triangleq [g(i+\epsilon\cdot\xi_i(S;\gamma);\beta)]_{i\in[l]} , \tag{3.18}$$

where $\xi_i(S;\gamma)$ denotes the i-th entry of the noise vector. This will allow every entry in the template score assignment to be adjusted by a phase shift of up to ϵ in magnitude. When $\epsilon = 0$, this space of template functions coincides with that of the exact periodic template model, thus encodes all theoretical optimal assignments. On the other hand, as ϵ increases, more template assignments are admitted, but the optimal density guarantee becomes less certain.

3.3.3 Specification of Distance Metric

As standard practice, we first consider as our objective the ℓ^2 distance, which is given by:

$$\Delta_{\ell^2}(\mathbf{f}(S;\alpha),\mathbf{g}(S;\beta)) \triangleq \sum_{i=1}^{l}(\mathbf{f}_i-\mathbf{g}_i)^2 , \tag{3.19}$$

where $\mathbf{f}_i$ and $\mathbf{g}_i$ are respectively the short-hands for the i^{th} entries of $\mathbf{f}(S;\alpha)$ and $\mathbf{g}(S;\beta)$. When the positional noise model is used, we assume that β also incorporates the noise parameter γ.

This metric, however, places a strict matching objective at all positions along $\mathbf{f}$ and $\mathbf{g}$, which is often unnecessary. We instead argue that it is sufficient to ensure that the k-mers that are likely to be selected (as suggested by the template assignment) are assigned lowest scores. Enforcing a perfect matching will take away the degrees of freedom needed for the proxy objective to satisfy the constraints implied by PRIORITYNET (i.e., a k-mer has to be assigned the same score at all of its occurrences).

As such, we construct an alternative distance metric that: (a) prioritizes matching positions around the window-minima of $\mathbf{g}$; and (b) allows flexible assignment at other positions to admit more solutions that meet the consistency requirement. To accomplish these design goals, we

propose the following asymmetrical distance metric:

$$\Delta_{\mathcal{DM}}(\mathbf{f}(S;\alpha),\mathbf{g}(S;\beta)) \triangleq \sum_{i=1}^{L_k}\Big[(1-\mathbf{g}_i)\cdot(\mathbf{f}_i-\mathbf{g}_i)^2+\lambda\cdot(1-\mathbf{f}_i)^2\Big]\,. \tag{3.20}$$

Specifically, the intuition behind the first component $(1-\mathbf{g}_i)\cdot(\mathbf{f}_i-\mathbf{g}_i)^2$ in the summation is to weight each position-wise matching term $(\mathbf{f}_i-\mathbf{g}_i)^2$ by its corresponding template score. The weight term $1-\mathbf{g}_i$ implies stronger matching preference around the minima of $\mathbf{g}$ where the template scores $\mathbf{g}_i$ are low; and vice-versa weaker matching preference at other locations where $\mathbf{g}_i$ are high. The second component $\lambda\cdot(1-\mathbf{f}_i)^2$, on the other hand, encourages PRIORITYNET to maximize its output scores whenever possible, which prevents the system from settling for a trivial solution where both $\mathbf{f}$ and $\mathbf{g}$ are squashed to zero. The trade-off between these two components is controlled by the magnitude of the hyper-parameter λ. Finally, we confirm that this distance metric is fully differentiable with respect to α,β, hence can be efficiently optimized using gradient-based techniques. The parameter gradients are given by:

$$\begin{aligned}\frac{\partial}{\partial\alpha}\Delta_{\mathcal{DM}}(\mathbf{f},\mathbf{g}) &= \sum_{i=1}^{l} a_i\cdot\frac{\partial}{\partial\alpha}\mathbf{f}_i\,,\\ \frac{\partial}{\partial\beta}\Delta_{\mathcal{DM}}(\mathbf{f},\mathbf{g}) &= \sum_{i=1}^{l} b_i\cdot\frac{\partial}{\partial\beta}\mathbf{g}_i\,,\end{aligned} \tag{3.21}$$

where the respective constants are derived as follows:

$$\begin{aligned}a_i &= 2\cdot(1-\mathbf{g}_i)\cdot(\mathbf{f}_i-\mathbf{g}_i)+2\lambda\cdot(\mathbf{f}_i-1)\,,\\ b_i &= -2\cdot(1-\mathbf{g}_i)\cdot(\mathbf{f}_i-\mathbf{g}_i)-(\mathbf{f}_i-\mathbf{g}_i)^2\,.\end{aligned} \tag{3.22}$$

3.3.4 Empirical study

We implement our method using PyTorch and deploy all experiments on a RTX-2060 GPU. Each training epoch computes a loss value that is averaged over $N=10$ randomly sampled subsequences of length $l=500\times(w+k)$. We set $\lambda=1$ and use architectures of PRIORITYNET and TEMPLATENET as given in Fig. 3.1 and Section 3.3.2 respectively. Network weights are optimized using the ADAM optimizer [54] with learning rate $\eta=5e^{-3}$.

Comparison baselines

We compare DEEPMINIMIZER with the following benchmarks: (a) random minimizer baseline; (b) Miniception [110]; (c) PASHA [26]; and (d) PolarSet Minimizer [111]. Among these methods, (d) is a sequence-specific minimizer scheme. For each method, we measure the density factor $D(S;\cdot) \triangleq (w+1)D(S;\cdot)$ to align with the convention of previous work (i.e., the theoretical lower bound on density factor is thus $D(S;\cdot) \geq 1+1/w$). Our empirical result is obtained on different segments of the human reference genome: (a) chromosome 1 (CHR1); (b) chromosome X (CHRX); (c) the centromere region of chromosome X [74] (which we denote by CHRXC); and (d) the full genome (HG38). We used lexicographic ordering for PASHA as suggested by Zheng et al. [110]. Random ordering is used to rank k-mers within the UHS for Miniception, and outside the layered sets for PolarSet. In most settings, we employ the Ensemble template model (Section 3.3.2) with no positional phase-shift component (Section 3.3.2) for DEEPMINIMIZER. However, for scenarios with large w values, we demonstrate that the Fourier template model with positional phase-shift is able to achieve better performance (Section 3.3.4)

Visualizing the mechanism of DEEPMINIMIZER

First, we show the transformation of the priority scores assigned by SCORENET and TEMPLATENET over 600 training epochs. Fig. 3.2 plots the outputs of these networks evaluated on positions 500 to 1000 of CHRXC, and their corresponding locations of sampled k-mers. As a practical implementation, we slightly modify the minimizer definition and use the MAXPOOL operator to select window maxima as minimizer locations (instead of window minima). Thus, we expect the sampled locations in Fig. 3.2 to coincide with the peaks of the priority scores (instead of the troughs). We also note that to accommodate this implementation, every relevant term in the DEEPMINIMIZER objective has been properly negated.

Initially, the PRIORITYNET assignment resembles that of a random minimizer and yields a density of 2.05. This agrees with previous theoretical result establishing that the expected den-

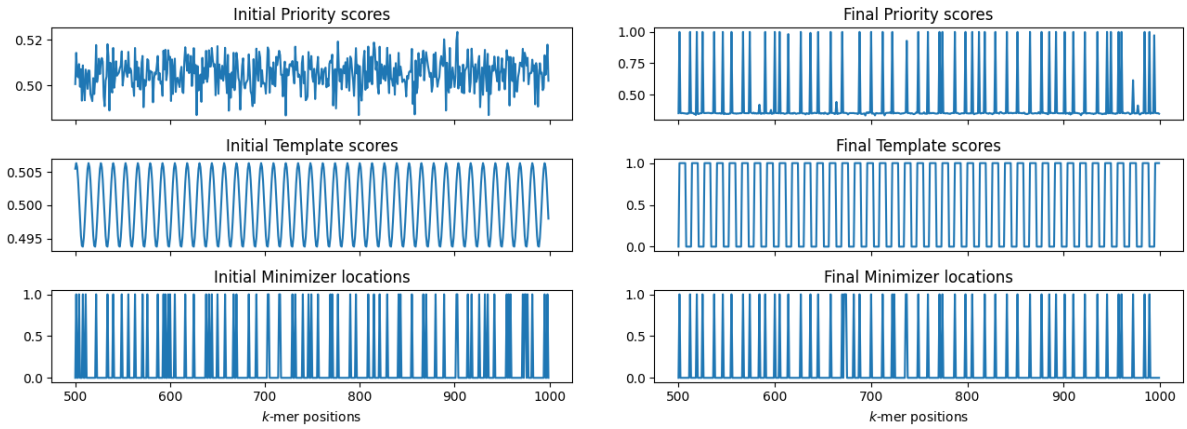


Figure 3.2: Visualization of PRIORITYNET and TEMPLATENET score assignments on positions $500 - 1000$ of CHRXC with $w = 13$, $k = 8$. Left: Initial assignments; Right: Final assignments after 600 training epochs. The bottom plots show corresponding locations of sampled k-mers: a value of 1 means selected, and 0 otherwise.

sity of a random minimizer is 2.0. After 600 training epochs, the final template score assignment converges to a different phase shift than its initial assignment, but its period remains the same. Simultaneously, PRIORITYNET learns to match its output to this template, hence induces a visibly sparser sketch with a density of 1.39. This result illustrates the negotiating process of our method to find a consensus score assignment.

Convergence of our proxy objective

We further demonstrate that our proxy objective meaningfully improves minimizer performance as it is optimized. The first two columns of Fig. 3.3 show the best density factors achieved by our method over 600 epochs on two scenarios: (a) varying k with fixed w; and (b) varying w with fixed k. The experiment is repeated on CHRXC and HG38. In each scenario, DEEPMINIMIZER starts with density $D \simeq 2.0$, which is comparable to the random minimizer baseline. We observe steady decrease of D over the first 300 epochs before reaching convergence, where total reduction ranges from $11 - 23\%$.

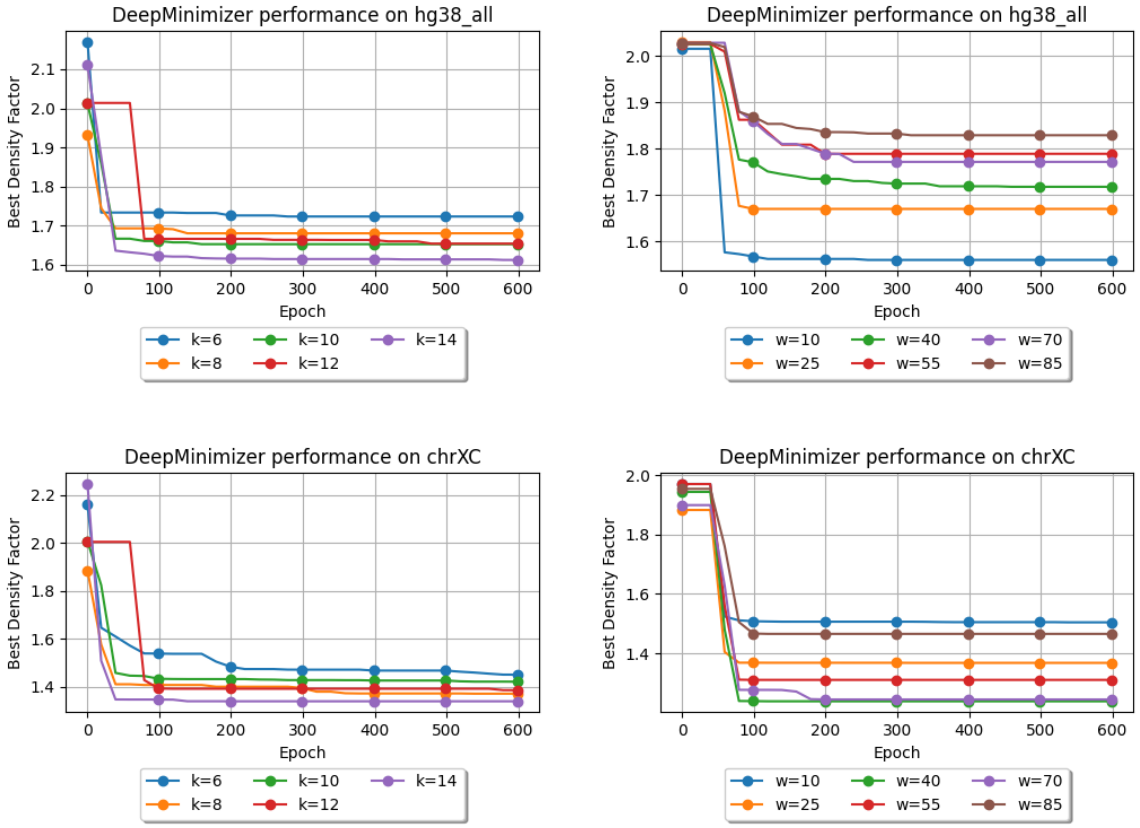


Figure 3.3: Best density factors obtained by DEEPMINIMIZER on HG38, CHRXC over 600 training epochs. Left: fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; Right: fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$.

Generally, larger k values lead to better performance improvement at convergence. This is expected since longer k-mers are more likely to occur uniquely in the target sequence, which makes it easier for a minimizer to achieve sparse sampling. In fact, previous results have shown that when k is much smaller than $\log w$, no minimizer will be able to achieve the theoretical lower-bound $D = 1/w$ [110]. On the other hand, larger w values lead to smaller improvements and generally slower convergence. This is because our ensemble parameterization of TEMPLATENET scales with the window size w and becomes more complicated to optimize as w increases.

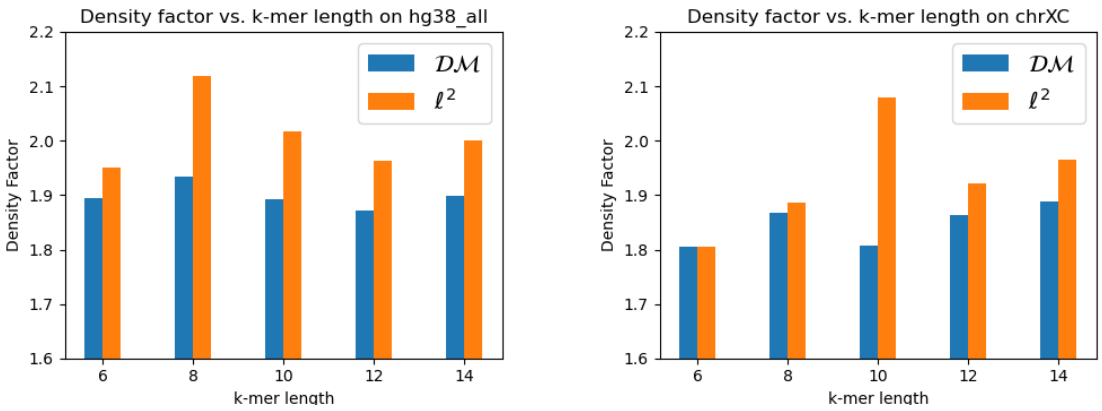


Figure 3.4: Comparing best density factors obtained by DEEPMINIMIZER with Δ_{ℓ^2} and $\Delta_{\mathcal{DM}}$ on HG38 (left) and CHRXC (right) over 600 training epochs.

Evaluating our proposed distance metric

Fig. 3.4 shows the density factors achieved by our DEEPMINIMIZER method, respectively specified by the proposed distance metric $\Delta_{\mathcal{DM}}$ in Eq. 3.20 and Δ_{ℓ^2} distance. Here, we fix $w = 13$ and vary $k \in \{6, 8, 10, 12, 14\}$. We observe that with the Δ_{ℓ^2} distance, we obtain performance similar to a random minimizer in most cases. On the other hand, with our divergence function, DEEPMINIMIZER obtains significantly lower densities, which confirms the intuition in Section 3.3.3.

Comparing against other minimizer optimization methods

We show the performance of DEEPMINIMIZER compared to other benchmark methods. In this experiment, DEEPMINIMIZER is trained for 600 epochs with ensemble TEMPLATENET and no positional phase-shift. Fig. 3.5 and Fig. 3.6 shows the final density factors achieved by all methods, again on two comparison scenarios: (a) fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; and (b) fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$. DEEPMINIMIZER consistently achieves better performance compared to *non-sequence-specific* minimizers (i.e., PASHA, Miniception) on all settings. We observe up to 40% reduction of density factor (e.g., on CHRXC, $w = 70$, $k = 14$), which clearly demonstrates the ability of DEEPMINIMIZER to exploit *sequence-specific* infor-

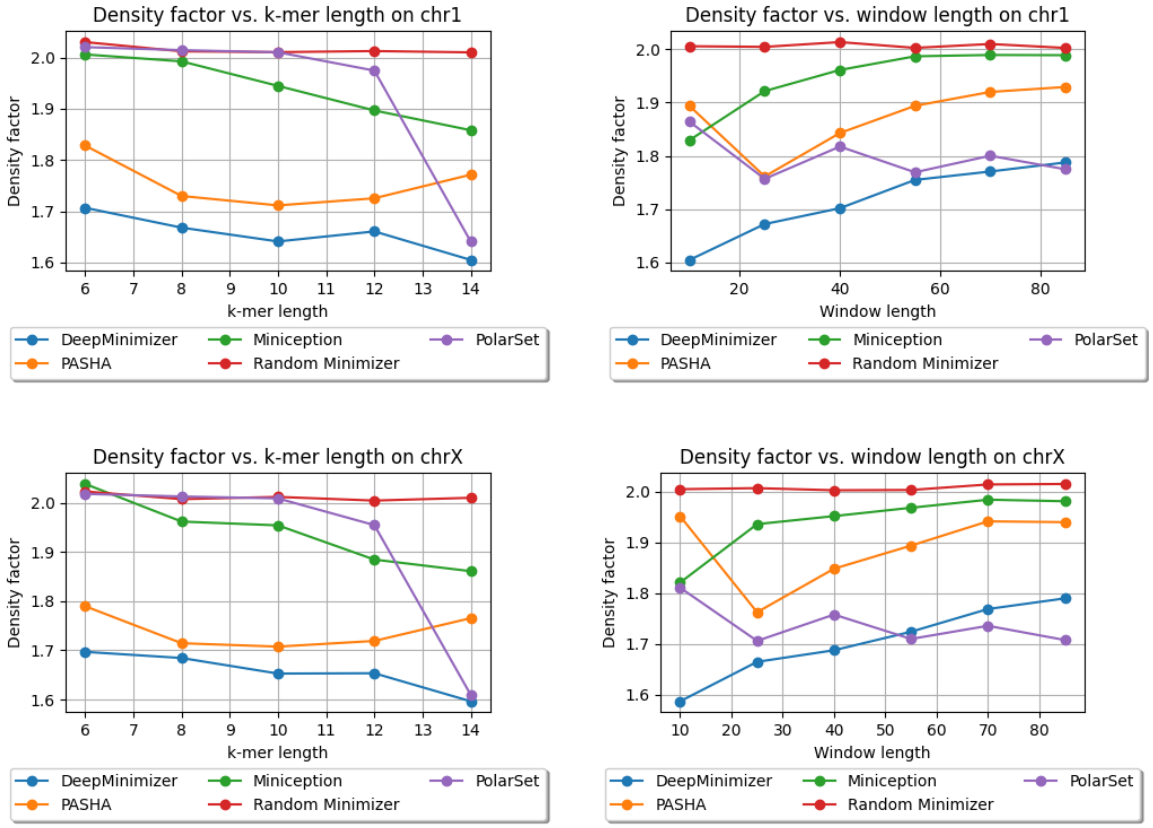


Figure 3.5: Density factors obtained by DEEPMINIMIZER (600 training epochs), Random Minimizer, PASHA, Miniception and PolarSet on CHR1, CHRX. Left: fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; Right: fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$.

mation. Furthermore, we also observe that DEEPMINIMIZER outperforms our *sequence-specific* competitor, PolarSet, in a majority of settings. The improvements over PolarSet are especially pronounced for smaller k values, which are known harder tasks for minimizers [110]. On larger w values, our method performs slightly worse than PolarSet in some settings. This is likely due to the added complexity of optimizing TEMPLATENET, as described in convergence ablation study of our method.

Notably, the centromere region of chromosome X (i.e., CHRXC) contains highly repetitive subsequences [32] and has been shown to hamper performance of PolarSet [111]. Fig. 3.6 shows that PolarSet and the UHS-based methods perform similarly to a random minimizer, whereas

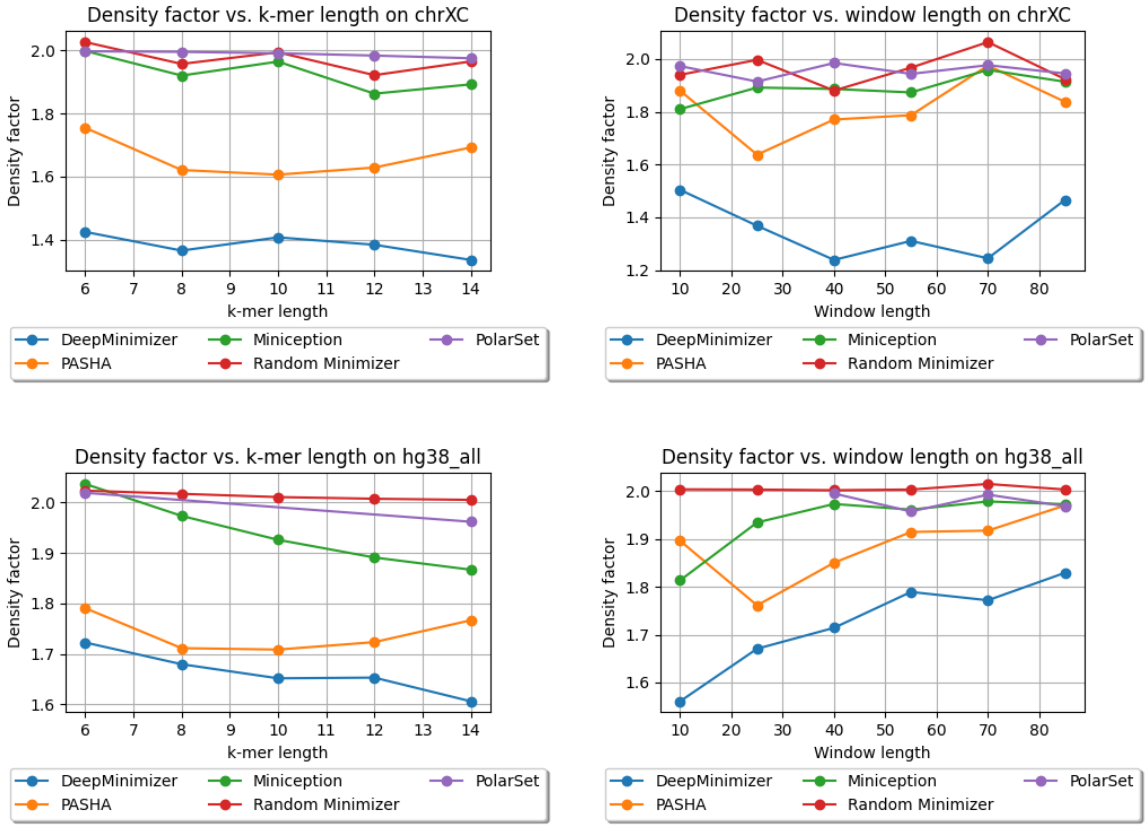


Figure 3.6: Density factors obtained by DEEPMINIMIZER (600 training epochs), Random Minimizer, PASHA, Miniception and PolarSet on CHRXC, HG38. Left: fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; Right: fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$.

our method is consistently better. Moreover, we observe that DEEPMINIMIZER obtains near-optimal densities with CHRXC on several settings. For example, we achieved $D = 1.22$ when $k = 14$, $w \in \{40, 70\}$, which is significantly better than the results on CHR1 and CHRX. This suggests that CHRXC is not necessarily more difficult to sketch, but rather good sketches have been excluded by the UHS and polar set reparameterizations, which is not the case with our framework.

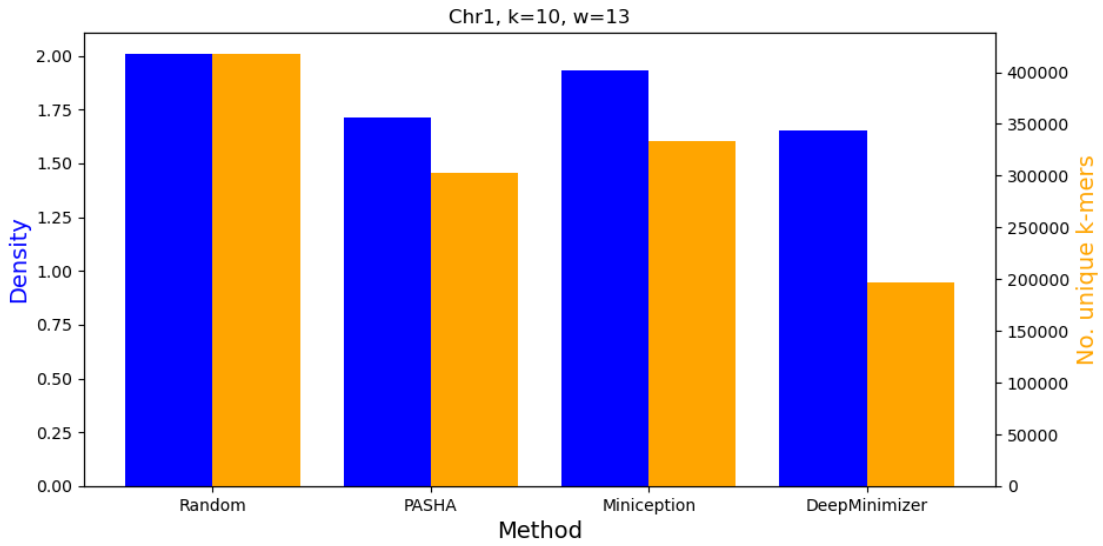


Figure 3.7: Comparing density and number of unique k-mers in the minimizer sets obtained by various benchmarks on CHR1 with $k = 10$ and $w = 13$.

Number of unique k-mers in the final minimizer set

This section investigates the numbers of unique k-mers in the final minimizer sets obtained by random ordering, PASHA, Miniception and DeepMinimizer. On CHR1, with $k = 10$ and $w = 13$, Fig 3.7 shows that the density factors and numbers of unique k-mers obtained by each method are strongly correlated. This agrees with the intuition of many other minimizer methods that a small set of high priority k-mers (e.g., a small UHS in the case of PASHA and Miniception) tends to induce a low density sketch on the target sequence. This observation is also expected since the 10-mer distribution of CHR1 is fairly similar to that of a random sequence, which aligns with the premise of most UHS-based minimizer theories.

However, on the chromosome region of CHRX, which contains many highly repetitive subsequences, Fig. 3.8 shows that in order to achieve the best density (i.e., $D = 1.526$), DEEPMINIMIZER actually had to pick more high priority k-mers, not fewer. This interestingly demonstrates that minimizing the size of the UHS is not always a desirable surrogate objective on certain specific sequences, hence asserts the need for a robust sequence-specific optimizer.

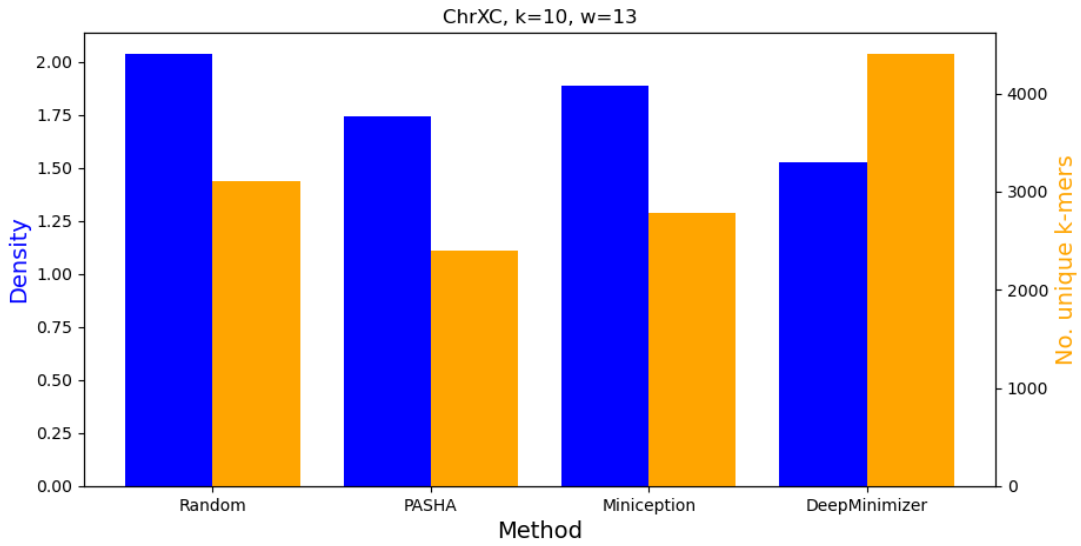


Figure 3.8: Comparing density and number of unique k-mers in the minimizer sets obtained by various benchmarks on CHRXC with $k = 10$ and $w = 13$.

Comparing template models on large window values

We investigate the performance of DEEPMINIMIZER on large window size with different template models. Particularly, we fixed $k = 20, w = 100$ and compare the best density factor obtained by DEEPMINIMIZER over 1200 training epochs using the ensemble template model (Section 3.3.2) and the truncated Fourier series template model (Section 3.3.2). We further pair each template model with a positional phase-shift component (Section 3.3.2), with $\epsilon \in \{0.0, 1.0, 10.0\}$. We note that in each case, $\epsilon = 0.0$ corresponds to the original template model.

Fig. 3.9 shows the respective loss and density factor over 1200 training epochs of these template models. First, we observe that in all models, the loss values correlate positively with the corresponding density factor. Generally, as the DEEPMINIMIZER loss decreases, the induced minimizer scheme also yields lower density factor on the input sequence, which suggests that our loss function is a good surrogate for the discrete density objective.

Furthermore, we observe that among variants of the Fourier template model, both $\epsilon = 1.0$ and $\epsilon = 10.0$ perform significantly better than $\epsilon = 0.0$. This is most likely because adding local

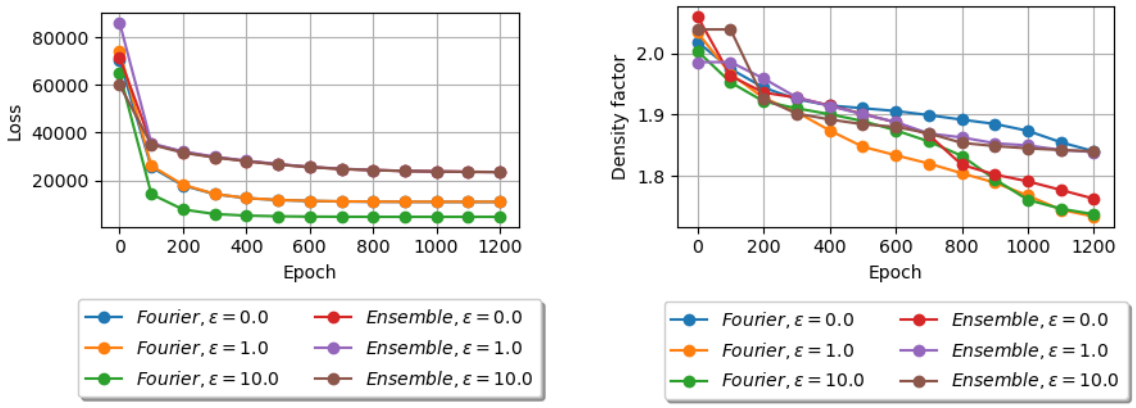


Figure 3.9: Comparing loss (left) and best density obtained (right) over 1200 training epochs on CHR1 between ensemble and truncated Fourier series template models. Each template model is paired with a positional phase-shift component with $\epsilon \in \{0.0, 1.0, 10.0\}$.

phase perturbations indeed allows TEMPLATENET to encode more realistic near-optimal score assignments. In contrast, among variants of the ensemble template model, $\epsilon = 0.0$ performs the best. This is most likely because the ensemble model has already accounted for all possible integer phase-shifts. As such, adding noisy phase perturbations with magnitude greater than 1.0 will negatively affect the convergence of DEEPMINIMIZER.

Finally, pairing Fourier template model with a positional phase-shift component of magnitude $\epsilon = 1.0$ achieves the best performance out of all variants. This aligns with our intuition in Section 3.3.2 regarding the trade-off between the certainty of Proposition 1 and the expressiveness of the admitted set of template score assignments.

Runtime performance

Finally, we confirm that DEEPMINIMIZER runs efficiently with GPU computing. In all of our experiments, each training epoch takes approximately 30 seconds to 2 minutes, depending on the choice of k and w, which controls the batch size. Performance evaluation takes between several minutes (CHRXC) to 1 hour (HG38), depending on the length of the target sequence. Generally,

our method is cost-efficient without frequent evaluations. Our most cost-intensive experiment (i.e., convergence ablation study on HG38) requires a full-sequence evaluation every 20 epochs over 600 epochs, thus takes approximately 2 days to complete. This is faster than PolarSet, which has a theoretical runtime of $\mathcal{O}(n^2)$ and takes several days to run with HG38. We note that in real applications, we only have to evaluate once by the end of the training loop, which is much faster compared to PolarSet, whose running time above only involves building the minimizer scheme.

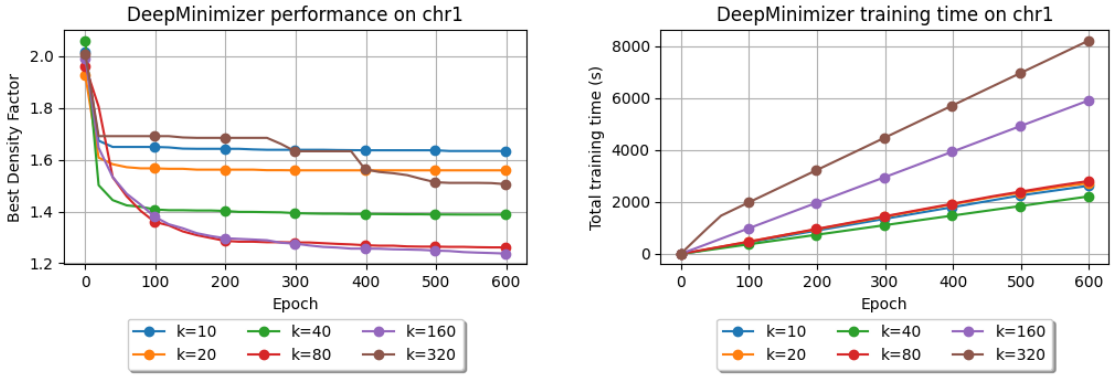


Figure 3.10: Best density obtained (left) and runtime (right) of DEEPMINIMIZER for $w = 13$ and $k \in \{10, 20, 40, 80, 160, 320\}$ on CHR1.

Fig. 3.10 (right) measures runtime (in seconds) of DEEPMINIMIZER on CHR1 over 600 epochs. Larger k values require PRIORITYNET to have more parameters. We expect running time for $k = 40, 80, 160, 320$ to increase in the same order. For $k = 10$ and 20, however, the running times are approximately the same as $k = 80$. We note that a smaller k value means there are more k-mers in the same sequence. As such, even though PRIORITYNET is more compact for these values of k, we will incur some overhead from querying it more often. For completeness, we also show the corresponding density performance plot in Fig. 3.10 (left), which confirms that our model converges well even for large k.

3.4 GSS optimization of the masked minimizer generalization

As an alternative to minimizers, the syncmer sketching scheme was recently proposed by Edgar [24] to construct sequence sketches that have high conservation (Definition 6). While the minimizer sampling rule is dependent on other k-mers in the same context window, the syncmer sampling rule trades off this window sampling mechanism for other useful properties, such as better robustness when sketching homologous sequences [24, 90]. In particular, k-mer sampling syncmer schemes are derived from s-mer orderings, where $s < k$. Let k_s be the number of s-mers in each k-mer. The *open-syncmer* variant samples every k-mer in which the lowest-ranked s-mer is found at the t^{th} offset position for some fixed $t \in [0, k_s - 1]$. The *closed-syncmer* sampling rule sets this offset position to be either the first or the last position.

The parameterized syncmer scheme [22] generalizes these syncmer variants using a subset parameter that encodes the selection rule. Specifically, given some subset $v \subseteq [0, k_s - 1]$, a v-parameterized syncmer scheme samples every k-mer in which the lowest-ranked s-mer is found at *some offset position* in v. This flexible encoding of sampling rules offers a practical handle on the performance of syncmers, where subsets that neither correspond to open-syncmer (i.e., $v = \{t\}$) nor closed-syncmer (i.e., $v = \{0, k_s - 1\}$) have been shown to outperform both original variants [22].

The major argument for choosing syncmers over minimizers is that high conservation is preferable when comparing sequences that might have diverged due to mutations and/or sequencing error. Shaw and Yu [90] subsequently demonstrated that syncmers indeed have better expected conservation than minimizers when both the input sequence and the ordering parameter are randomly drawn from uniform distributions. However, unlike minimizers, syncmer sketches are not guaranteed to sufficiently cover all regions of the target sequence. Thus, to fairly assess the quality of a general sketching scheme, we must additionally consider the *coverage* metric (Definition 7) that measures how sketched k-mers are spread out across the input sequence. To this end, we show that these three metrics are often adversarial to one another, and consequently

propose a more holistic *generalized sketch score* (GSS) to evaluate sketching performance (Section 3.4.1).

Previous studies have established theoretical guarantees for the expected density of minimizers [70, 89] and syncmers [24] with uniformly random input sequences. The majority of sketch construction methods derived from these results thus only learn the ordering once and reuse it for all applications. However, when dealing with scenarios involving multiple query sequences being aligned against a single reference string, such as genome assembly, it is often more desirable to have an ordering that is optimally configured based on the reference. For instance, both Zheng et al. [111] and the DEEPMINIMIZER method are practical algorithms to optimize the k-mer ordering in the minimizer method. These studies have demonstrated that sequence-specific minimizer sketches generally achieve much lower density compared to non-optimized minimizer sketches.

Nonetheless, these optimization methods cannot be directly applied to configure low-density syncmer sketches, because they explicitly leverage the minimizer window sampling mechanism to construct their respective learning objectives. For example, the polar set method adopts a heuristic that selects as many k-mers as possible from the set of k-mers that are w (i.e., window size) bases apart [111], whereas the DEEPMINIMIZER method introduced in this thesis constructs a sinusoidal template function with period w to guide optimization. In addition, syncmers have no minimum density guarantee, unlike minimizers that derive this property from the window sampling mechanism. As such, optimizing the syncmer method for a specific sequence can potentially result in a vacuous sketch with zero coverage. Finally, extending previous density optimization methods to account for the conservation metric is also challenging, as conservation and density are adversarial metrics (Section 3.4.1).

To address these challenges, we adapt the parameterized syncmer framework [22] such that the pattern-aware sampling rules are applied in conjunction with the window sampling rule of minimizers. We call this adaptation masked minimizers. Specifically, given a subset v (or equiv-

alently a binary mask variable in our formulation), the masked minimizer scheme selects all minimizers that are found at some offset position in v (with respect to the windows they minimizer). Similar to the parameterized syncmer framework, the pattern-aware sampling rules give masked minimizers the ability to balance the trade-off between density, conservation, and coverage. However, our formulation differs from that of Dutta et al. [22] since the selection of a masked minimizer depends on k-mers around it, whereas a parameterized syncmer is selected in a context-free manner. This distinction is identical to how minimizers and syncmers differ and allows us to leverage and extend density optimization algorithms developed for minimizers (Section 3.4.4).

In particular, we develop a sequence-specific optimization algorithm for masked minimizers that extends the DEEPMINIMIZER method. Our algorithm adopts a bi-level learning framework that alternates between pruning the mask variable and learning the k-mer ordering. Given a fixed mask, the inner loop optimizes the ordering via combining two differentiable objectives that respectively surrogate the density and conservation of the masked minimizer scheme. Alternately, the outer loop searches for the optimal mask via greedily pruning its set bits, suggesting pruned candidates to the inner loop, and selecting one that yields the best metric gain.

We show that the optimized masked minimizer sketch of various human and bacterial genomes achieves better GSS than previous optimization approaches, such as MINICEPTION [110], PASHA [26] and DEEPMINIMIZER. We also discover a specific class of complement mask patterns (i.e., masks that include most offset positions except one) that combines desirable properties from minimizers (i.e., high coverage) and open-syncmers (i.e., tolerance to low-complexity sequences).

3.4.1 Generalized sketch score

It is straight-forward to see that the conservation metric (Definition 6) is naturally upper-bounded by the density metric (Definition 5) with respect to any arbitrary sketching scheme $\mathcal{X}$:

$$\begin{aligned} C(S;\mathcal{X},\theta) &= \frac{1}{L}\mathbb{E}_{S'\sim p_S}\left|\mathcal{X}(S;\theta)\cap\mathcal{X}(S';\theta)\right| \\ &\leq \frac{1}{L}\mathbb{E}_{S'\sim p_S}\left|\mathcal{X}(S;\theta)\right| \\ &= \frac{1}{L}\left|\mathcal{X}(S;\theta)\right| \;=\; D(S;\mathcal{X},\theta)\;. \end{aligned} \tag{3.23}$$

This implies that these metrics are mutually conflicting, and since both of them do not account for coverage (Definition 7), neither can sufficiently measure the quality of a sketch. This motivates us to construct a more holistic sketching metric, which we call the *generalized sketch score* (GSS), to evaluate the performance of sketching schemes. Intuitively, the GSS metric encourages striking a balance between high conservation, low density, and high coverage. This is achieved by measuring the trade-off ratio between conservation/density, and normalizing this value by the w-coverage score of the sketch:

$$G_w(S;\mathcal{X},\theta) \;\triangleq\; \frac{C(S;\mathcal{X},\theta)}{D(S;\mathcal{X},\theta)} \times V_w(S;\mathcal{X},\theta). \tag{3.24}$$

As a consequence of Definition 7 and Eq. (3.23), G_w is guaranteed to be in $[0,1]$.

3.4.2 Masked minimizers

While sequence-specific optimization of minimizers with respect to the density metric has been addressed by Ekim et al. [26], Zheng et al. [110] and the DEEPMINIMIZER method presented earlier in this chapter, the same capability has not been developed for either open syncmers, parameterized syncmers, or for any other metrics than density. To overcome this challenge, our goal is to extend the minimizer method with pattern-based sampling rules similar to that of parameterized syncmers. This extension allows us to incorporate desirable properties from the syncmer family, yet fully retain access to density optimization algorithms developed for

minimizers. As a result, we introduce the masked minimizer scheme specified by the tuple (w, k, v, π). The parameters w, k, π maintain the same definition as in the original minimizer scheme. Similar to parameterized syncmers [22], v denotes a subset of qualifying offsets (e.g., a binary mask) such that a minimizer is chosen only if its relative location in the window is within v. The masked minimizer sampling rule is:

$$\mathcal{V}(S; w, k, v, \pi) \triangleq \{i + m_f(i, w) \mid m_f(i, w) \in v\}_{i \in [1, L_{w_k}]} . \tag{3.25}$$

3.4.3 Relating density and conservation metrics of masked minimizers

This section provides an analysis of the change in performance of the masked minimizer scheme as v varies in the power set of $[0, w-1]$. In particular, we ask whether conservation/density will improve with more or fewer offset locations in the qualifying subset v? Specifically, let $\theta = (w, k, v, \pi)$ and $\theta' = (w, k, v', \pi)$ be the parameters defining two masked minimizer schemes such that $v \subseteq v' \subseteq [0, w-1]$, our analysis seeks to bound their performance gap in terms of density and conservation metrics.

Proposition 2 *For any input sequence S and parameters θ, θ' defined above, we have $\mathcal{V}(S; \theta) \subseteq \mathcal{V}(S; \theta')$.*

Proof: Let $i \in \mathcal{V}(S; \theta)$. By definition of the masked minimizer sampling rule, there exists $j \in [1, L_{w_k}]$ such that $j + m_f(j, w) = i$ and $m_f(j, w) \in v$. Since $v \subseteq v'$, we also have $m_f(j, w) \in v'$, which implies $i \in \mathcal{V}(S; \theta')$, again by definition of the masked minimizer rule. Therefore, $\mathcal{V}(S; \theta) \subseteq \mathcal{V}(S; \theta')$. □

Corrolary 1 (Density gap). For any input sequence S and parameters θ, θ' defined above, we have $D(S; \mathcal{V}, \theta) \leq D(S; \mathcal{V}, \theta')$.

Proof: By definition of density:

$$D(S; \mathcal{V}, \theta) = \frac{|\mathcal{V}(S; \theta)|}{L_k} \leq \frac{|\mathcal{V}(S; \theta')|}{L_k} = D(S; \mathcal{V}, \theta') , \tag{3.26}$$

where the inequality follows directly from Proposition 2. □

Corollary 2 (Conservation gap). For any input sequence S and parameters θ, θ' defined above, we have $C(S; \mathcal{V}, \theta) \leq C(S; \mathcal{V}, \theta')$.

Proof: Let S' be a homologous copy of S obtained through simulating base substitutions. We additionally define $\alpha_i(S', \theta^\dagger) \triangleq \mathbb{I}(i \in \mathcal{V}(S; \theta^\dagger) \cap \mathcal{V}(S'; \theta^\dagger))$, which indicates the event that i is preserved in both $\mathcal{V}(S; \theta^\dagger)$ and $\mathcal{V}(S; \theta^\dagger)$ for some arbitrary sampling parameter tuple $\theta^\dagger$. We then have the following:

$$\begin{aligned} \alpha_i(S', \theta) &= \mathbb{I}(i \in \mathcal{V}(S; \theta)) \times \mathbb{I}(i \in \mathcal{V}(S'; \theta)) \\ &\leq \mathbb{I}(i \in \mathcal{V}(S; \theta')) \times \mathbb{I}(i \in \mathcal{V}(S'; \theta')) \\ &= \alpha_i(S', \theta') \,, \end{aligned} \tag{3.27}$$

where the inequality follows from Proposition 2, and the fact that the indicator variables take values in $\{0, 1\}$. We now bound the conservation gap as follows:

$$\begin{aligned} C(S; \mathcal{V}, \theta) - C(S; \mathcal{V}, \theta') &= \mathbb{E}_{S'} \frac{|\mathcal{V}(S; \theta) \cap \mathcal{V}(S'; \theta)| - |\mathcal{V}(S; \theta') \cap \mathcal{V}(S'; \theta')|}{L_k} \\ &= \mathbb{E}_{S'} \frac{\sum_{i=1}^{L_k} \alpha_i(S', \theta) - \alpha_i(S', \theta')}{L_k} \;\leq\; 0 \,, \end{aligned} \tag{3.28}$$

where the inequality follows from Eq. (3.27) and linearity of expectation. Rearranging the above result concludes the proof. □

These results imply that any masked minimizer scheme can improve conservation by adding more locations to its qualifying subset, or improve density by removing locations. However, as density upper-bounds conservation (Section 3.4.1), it is difficult to simultaneously improve both metrics by varying the mask, and hence it is necessary to formulate the optimization in terms of their trade-off ratio (e.g., the GSS metric), and with respect to the mask variable.

3.4.4 Optimizing masked minimizers

Algorithm 1 Masked minimizer optimization

best-gss $\leftarrow 0$

mask $\leftarrow \mathbf{1}^w$

gss $\leftarrow \text{eval}\left(\underset{f,g}{\text{argmin}}\ \mathcal{L}_{gss}(\text{seq}; \text{mask})\right)$ {Eq. 3.30}

while gss $>$ best-gss **and** not-empty(mask) **do**

  best-gss $\leftarrow$ gss

  best-mask $\leftarrow$ mask

  for offset $\in$ mask **do**

    trial-gss $\leftarrow \text{eval}\left(\underset{f,g}{\text{argmin}}\ \mathcal{L}_{gss}(\text{seq}; \text{prune}(\text{mask}, \text{offset}))\right)$ {Eq. 3.30}

    if trial-gss $>$ gss **then**

      gss $\leftarrow$ trial-gss

      best-mask $\leftarrow \text{prune}(\text{mask}, \text{offset})$

  mask $\leftarrow$ best-mask

return $f, g,$ best-mask

To represent k-mer orderings, we employ a continuous scoring function $f : \Sigma^k \rightarrow [0, 1]$ similar to the DEEPMINIMIZER framework. Given a choice of w, k, we thus seek to optimize f and the mask parameter v of the masked minimizer scheme with respect to the GSS metric. To achieve this, we adopt a bi-level optimization framework, which iterates between: (a) taking gradient descent steps on the weights of f given a fixed v; and (b) greedily pruning v to improve GSS given an optimized f. The pseudocode of our framework is given in Algorithm 1.

Our greedy pruning step (outer loop) starts with the complete qualifying set $v = [0, w-1]$ and iteratively removes locations one by one from v to yield the best GSS improvement, given one full inner loop optimization of f. This outer loop terminates when no further improvement can be obtained or the mask is empty (i.e., $|v| = 0$). To address the inner loop optimization, we construct a differentiable loss function which extends the DEEPMINIMIZER algorithm to account for the conservation component of the GSS metric. Specifically, we augment the DEEPMINIMIZER loss

function with an auxiliary term that estimates the stability of the PRIORITYNET score assignment when subjected to random mutations. This loss function thus surrogates the trade-off between density and conservation, and it will implicitly allow us to maximize the GSS metric. We describe the components of our loss function as follows.

Density optimization. Similar to DEEPMINIMIZER, we employ a collaborating neural system comprising the PRIORITYNET and the TEMPLATENET to model a low density scoring function. Recall that these networks respectively ensure the validity of a minimizer scheme and the ideal low density. A low-density sketch thus can be viewed as a consensus score assignment $\mathbf{f} \triangleq [f(\kappa_i^k)]_{i\in[L_k]}$ that minimizes some distance metric Δ to an arbitrary template assignment $\mathbf{g} \triangleq [g(i)]_{i\in[L_k]}$ in the output space of the TEMPLATENET:

$$\Delta(\mathbf{f}, \mathbf{g}; v) \triangleq \lambda \sum_{i=1}^{L_k} (1 - \mathbf{f}_i)^2 + \sum_{i=1}^{L_{w_k}} \sum_{j\in v} (1 - \mathbf{g}_{i+j})(\mathbf{f}_{i+j} - \mathbf{g}_{i+j})^2 \,, \tag{3.29}$$

where the first term $\lambda \sum_{i=1}^{L_k} (1 - \mathbf{f}_i)^2$ also exists in the formulation of the DEEPMINIMIZER distance metric $\Delta_{\mathcal{DM}}$, and serves as a regularization term that ensures $\mathbf{f}$ and $\mathbf{g}$ are not trivially set to 0. On the other hand, the second term differs from that of $\Delta_{\mathcal{DM}}$ by the inner summation over the offset positions in v, which is specific to the masked minimizer method. This sum represents an aggregation of weighted ℓ_2-distances over all (w, k)-windows of k-mer scores in $\mathbf{f}$ and $\mathbf{g}$. The weight at each k-mer location is therefore jointly determined by its template value (i.e., how likely it is that this position will contribute to the sketch) and whether it can be found in the qualifying subset of some window (i.e., how relevant this is position to the current sampling rule). Last, the parameter λ captures the trade-off between these two objective terms.

Conservation optimization. To further account for conservation, we extend the above distance metric with an additional objective to yield the final loss function $\mathcal{L}_{gss}$:

$$\mathcal{L}_{gss}(S; v) \triangleq \Delta(\mathbf{f}, \mathbf{g}; v) + \frac{\lambda_c}{n} \sum_{t=1}^{n} \Delta(\mathbf{f}'_t, \mathbf{g}; v) \,, \tag{3.30}$$

where $\mathbf{f}'_{t=1\ldots n}$ denotes the PRIORITYNET score assignments corresponding to homologous copies

$S'_{t=1\ldots n}$ of S simulated by random base mutation. and λ_c balances between the density and conservation objectives. $\mathcal{L}_{gss}$ is optimized with respect to the combined parameters of f and g. The first term of $\mathcal{L}_{gss}$ is exactly the density loss described above. The second term, on the other hand, surrogates the conservation metric by estimating the expected Δ-distance from each $\mathbf{f}'_t$ to the template score $\mathbf{g}$. When this term is small, we intuitively expect that $\{\mathbf{f}'_t\}_{t=1\ldots n}$ are concentrated around $\mathbf{g}$, and by extension are also concentrated around $\mathbf{f}$, as $\mathbf{f}$ is brought close to $\mathbf{g}$ via minimizing the density term. Since $\mathbf{f}'_t$ and $\mathbf{f}$ respectively surrogates the sketches of S'_i and S, when this happens, the sketch of S is likely preserved across homologous sequences and yields high conservation.

3.4.5 Empirical study

We demonstrate the effectiveness of our optimization algorithm in learning high conservation, low density, and high coverage masked minimizer sketches. We also explore various ablation scenarios to confirm the practical usage of various specific masks (qualifying subsets).

Experimentation details

We compare the following baselines to construct the k-mer ordering for masked minimizers: (1) random ordering; (2) training with variants of our objective, including the previously introduced DEEPMINIMIZER loss function; (3) MINICEPTION [110]; and (4) PASHA [26]. All experiments are conducted on the human chromosome 1 (labelled CHR1); the centromere region of human chromosome X (labelled CHRXC); and several bacterial genomes that were previously used in Edgar [24] (labelled BTR1, BTR2, BTR3 and BTR4). The details of these sequences are given in Table 3.2.

We implement our method using PyTorch and deploy all experiments on a RTX-3080 GPU. Due to limited GPU memory, each training epoch only computes the loss on a randomly sampled batch of 32 substrings of length $\ell = 1500$ bases. The conservation component of $\mathcal{L}_{gss}$ is

averaged over 5 random mutations, simulated using a 10% base substitution rate. Evaluation of conservation is likewise obtained using 5 random mutations. Network weights are optimized using the ADAM optimizer [54] with default parameters. Our PyTorch implementation is available at `https://github.com/Kingsford-Group/maskedminimizer`.

Table 3.1: Descriptions and lengths of benchmark sequences

Label	Description (Assembly)	Length
CHRXC	Centromere region of human chromosome X [74]	3106132
CHR1	Human chromosome 1	233587144
BTR1	Blautia producta (GCA_004210255.1)	6354838
BTR2	Blautia hansenii DSM 20583 (GCF_002222595.2)	3065949
BTR3	[Clostridium] scindens (GCA_009684695.1)	3785527
BTR4	Blautia producta ATCC 27340 = DSM 2950 (GCA_010669205.1)	6197116

Adversarial relationship of density and conservation

This experiment demonstrates that density and conservation are indeed conflicting objectives and confirms our argument in Section 3.4.1. Specifically, we train two masked minimizers using the minimizer mask $v_m = [0, w-1]$ and the open-syncmer mask $v_o = \{w/2\}$ for $w = 7, k = 15$. We note that the masked minimizer scheme with v_o employs the window-based sampling mechanism, hence does not recover exactly the open-syncmer scheme (or equivalently the parameterized syncmer scheme with mask v_o), and only emulates its sampling pattern in the context of minimizers. We respectively denote these schemes by $\mathcal{M}$ and $\mathcal{O}_{w/2}$ and optimize them with three variants of our loss function:

- The extended DEEPMINIMIZER density loss for mask minimizers, given by $\mathcal{L}_{DM} \triangleq \Delta(\mathbf{f}, \mathbf{g})$.
- The conservation loss given by the second term in Eq. (3.30), $\mathcal{L}_{con} \triangleq \frac{1}{n}\sum_{t=1}^{n} \Delta(\mathbf{f}'_t, \mathbf{g})$.
- Our loss function $\mathcal{L}_{gss} \triangleq \mathcal{L}_{DM} + \lambda_c \mathcal{L}_{con}$ given in Eq. (3.30) with $\lambda_c = 1$.

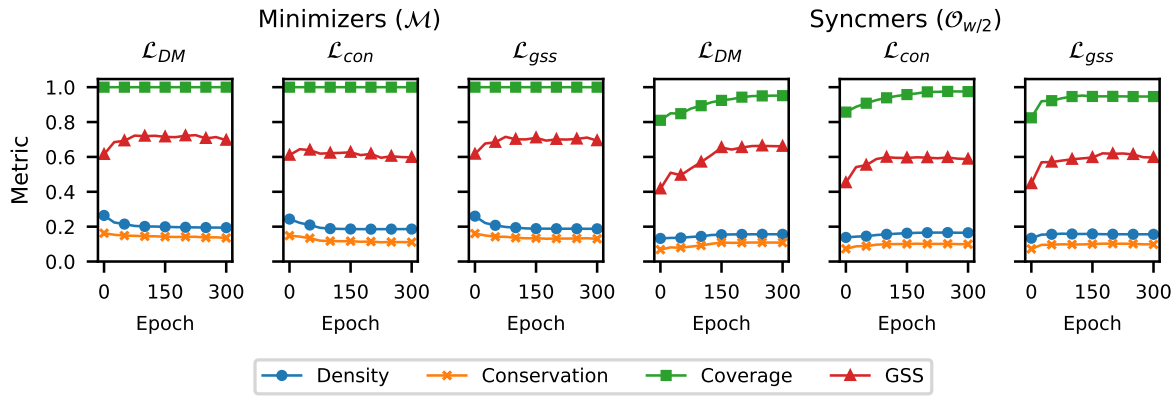
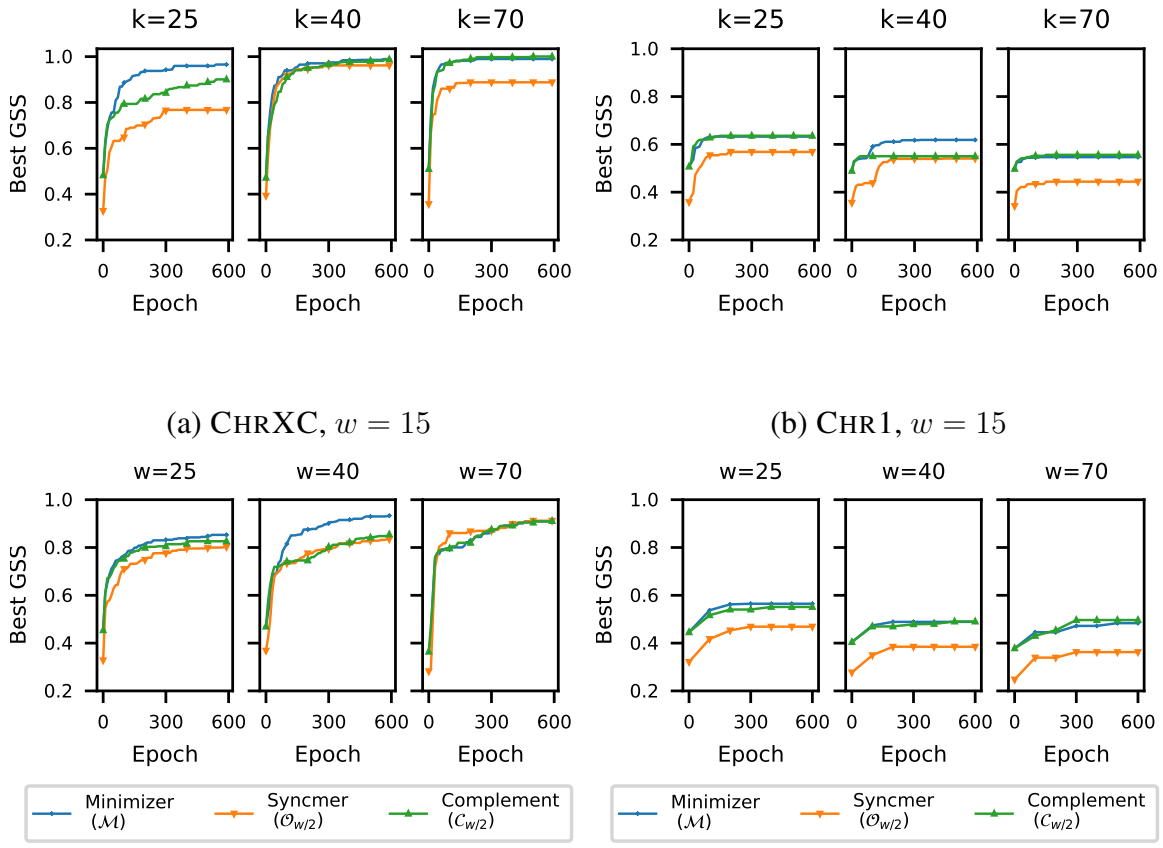


Figure 3.11: Comparing density, conservation, coverage and GSS vs. number of training epochs using different training losses and masks v on the bacterial genome BTR1.

Fig. 3.11 plots the density, conservation, coverage and GSS metrics on the sequence BTR1 across 300 training epochs for each loss function. As predicted in Section 3.4.1, we observe that the conservation metric is consistently upper-bounded by the density metric in all experiments. Additionally, we observe that neither the density nor conservation metric reflects the drop in coverage when moving from the minimizer mask $\mathcal{M}$ to the open-syncmer mask $\mathcal{O}_{w/2}$. The GSS metric, on the other hand, properly reflects this by applying a discount to the performance of the open-syncmer mask.

Training masked minimizers improves GSS

This section demonstrates that our loss function $\mathcal{L}_{gss}$ learns robustly and improves GSS in various settings of w, k, and different masks v. Specifically, we compare the minimizer mask (v_m) and the open-syncmer mask (v_o) defined above with the complement mask $v_c = v_m \setminus v_o$ that combines desirable properties from minimizer (e.g., high coverage) and open-syncmer mask (e.g., preventing repeated sampling in homopolymers). We denote the complement mask by $\mathcal{C}_{w/2}$. Again, we do not employ the outer loop of our algorithm to search for the optimal mask since it involves multiple inner iterations of training and cannot be plotted on the same scale with other benchmarks (e.g., each corresponds to a single inner loop with 600 epochs). The performance of



(a) CHRXC, $w = 15$

(b) CHR1, $w = 15$

(c) CHRXC, $k = 15$

(d) CHR1, $k = 15$

Figure 3.12: Comparing GSS of different masked minimizer variants vs. number of training epochs on CHRXC and CHR1.

this outer loop training will be demonstrated in the next experiment.

Fig. 3.12 plots the GSS of the masked minimizers $\mathcal{M}$, $\mathcal{O}_{w/2}$ and $\mathcal{C}_{w/2}$ over 600 training epochs in two settings: (1) $w = 15$ and $k \in [25, 40, 70]$; (2) $k = 15$ and $w \in [25, 40, 70]$. This experiment is repeated on two sequences, CHRXC and CHR1. All experiments show that GSS steadily increases over 600 training epochs by 1.5 to 5 times that of their initial random weights. The performance of the minimizer mask ($\mathcal{M}$) is highly similar to the complement mask ($\mathcal{C}_{w/2}$), except for $(w, k) = (15, 40)$ with CHR1 and $(w, k) = (15, 25), (40, 15)$ with CHRXC. This is expected because their masks only differ by one location. We further observe that both the minimizer mask ($\mathcal{M}$) and the complement mask ($\mathcal{C}_{w/2}$) outperform the open syncmer mask

($\mathcal{O}_{w/2}$) in most settings. This is most likely due to the worse coverage of open-syncmers, which has been previously observed in Fig. 3.11.

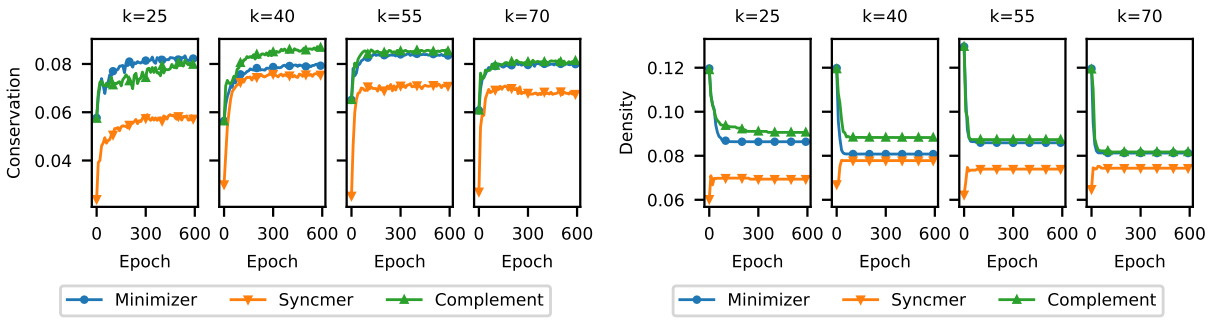


Figure 3.13: Comparing conservation and density metrics of different masked minimizers vs. number of training epochs on the CHRXC sequence with $w = 15$ and $k \in \{25, 40, 55, 70\}$.

Fig. 3.13 further demonstrates the individual effect of training the proposed loss $\mathcal{L}_{gss}$ on the conservation and density metrics. We observe that both the conservation and density of the open-syncmer scheme are upper-bounded by that of the minimizer scheme, which confirms the result of Corollary 1 and Corollary 2. We observe that $\mathcal{L}_{gss}$ improves conservation but worsens density for the open-syncmer scheme, which is similar to our first experiment. However, this is not the case for the minimizer and complement schemes, which obtain significant improvements in both metrics over 600 training epochs (although conservation is still bounded by density at any point during the training). This implies that our method has found a favorable trade-off between the two metrics, which in turn explains the sharper increases in GSS compared to that of syncmer across all experiments.

Comparing GSS of different training losses and masks

We demonstrate the importance of optimizing for the mask variable. Specifically, we compare the GSS performance among methods that optimize for the k-mer ordering alone with respect to some fixed mask, and our method (Algorithm 1) that jointly optimizes both variables. We

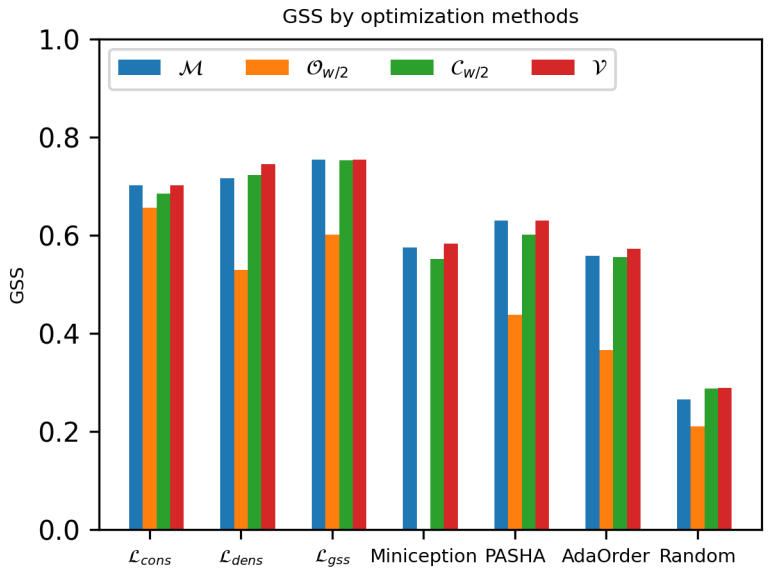


Figure 3.14: Comparing GSS of different masked minimizers using various optimization methods with $w = 10, k = 10$ on CHRXC. We respectively denote the minimizer mask, the open-syncmer mask, the complement mask and the optimized mask by $\mathcal{M}, \mathcal{O}_{w/2}, \mathcal{C}_{w/2}, \mathcal{V}$.

respectively denote the optimized mask and its induced masked minimizer scheme by v_* and $\mathcal{V}$. We benchmark the performance of this mask-optimized scheme against the minimizer $(\mathcal{M})$, open-syncmer $(\mathcal{O}_{w/2})$ and complement masks $(\mathcal{C}_{w/2})$ across various optimization strategies, including random orderings, PASHA [26], Miniception [110], AdaOrder [29], and the 3 differentiable loss functions previously introduced (i.e., $\mathcal{L}_{DM}$, $\mathcal{L}_{con}$, $\mathcal{L}_{gss}$). For random ordering and the UHS-based methods, which only select the ordering once, the inner-loop optimization is simply replaced by evaluating the GSS metric with respect to the current v. We repeat our experiment for $w = 10, k = 10$ (Fig. 3.14) and $w = 15, k = 10$ (Fig. 3.15).

Among different masks of the same optimization method, we observe that the optimized mask $\mathcal{V}$ achieves the best GSS most frequently (i.e., 13 out of 14 scenarios). Out of 13 occurrences, $\mathcal{V}$ recovers the same GSS as the minimizer mask $\mathcal{M}$ 5 times, and the same GSS as the complement mask $\mathcal{C}_{w/2}$ 3 times. The open-syncmer mask $\mathcal{O}_{w/2}$ only outperforms $\mathcal{V}$ one time on the random ordering baseline, with negligible margin. Interestingly, when combined with the MINICEPTION

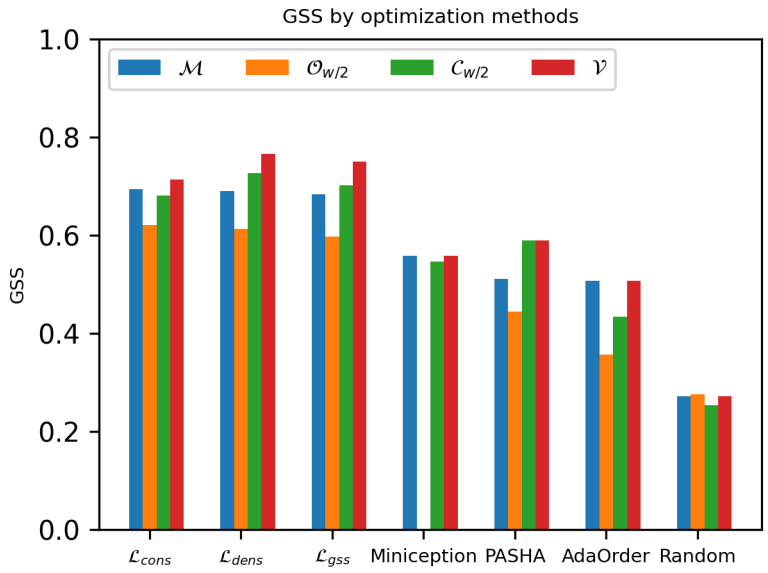


Figure 3.15: Comparing GSS of different masked minimizers using various optimization methods with $w = 15, k = 10$ on CHRXC. We respectively denote the minimizer mask, the open-syncmer mask, the complement mask and the optimized mask by $\mathcal{M}, \mathcal{O}_{w/2}, \mathcal{C}_{w/2}, \mathcal{V}$.

method, the open-syncmer mask yields 0.0 GSS, which suggests that there are no k-mers that can meet the sampling rule based on the ordering found by MINICEPTION.

Table 3.2 summarizes the result of the gradient-based methods on CHRXC for all combinations of $w \in \{10, 15, 20\}$ and $k \in \{10, 15\}$. Across 18 experiments (i.e., crossing 6 settings of (w, k) with 3 loss functions), the best GSS is achieved by the minimizer mask ($\mathcal{M}$) on 6 experiments, the open-syncmer mask ($\mathcal{O}_{w/2}$) on 1 experiment, and the complement mask ($\mathcal{C}_{w/2}$) on 4 experiments. Our optimized mask ($\mathcal{V}$) achieves the best GSS in 17 out of 18 experiments, including 10 experiments where v_* recovers either v_m, v_c or v_o; and 7 experiments where v_* is novel. Our loss function $\mathcal{L}_{gss}$ achieves the best GSS in 4 out of 6 combinations of (w, k).

Table 3.3 summarizes the result of PASHA [26], MINICEPTION [110] and the random ordering baseline for all combinations of $w \in \{10, 15, 20\}$ and $k \in \{10, 15\}$. Generally, PASHA and MINICEPTION outperform the random ordering baseline as expected. However, their performance is generally weaker than the gradient-based methods in Table 3.2 by a large margin.

Table 3.2: Comparing GSS (normalized to the scale of 0–100) of different masked minimizers with 3 different training losses across 6 settings of (w, k) on CHRXC. The best GSS observed for each combination of (w, k) and loss function is given in **bold**. The best GSS for each combination of (w, k) is further underlined.

	Conservation Loss ($\mathcal{L}_{con}$)				Density Loss ($\mathcal{L}_{DM}$)				Combined Loss ($\mathcal{L}_{gss}$)			
(w, k)	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$
$10, 10$	**70.1**	65.6	68.5	**70.1**	71.6	52.9	72.2	**74.5**	**75.4**	60.1	75.3	**75.4**
$10, 15$	**70.6**	65.0	69.6	**70.6**	76.8	65.6	**77.9**	**77.9**	71.9	56.6	70.4	**81.0**
$15, 10$	69.3	62.1	68.0	**71.3**	69.0	61.3	72.6	**76.5**	68.3	59.7	70.1	**75.0**
$15, 15$	71.5	66.7	71.0	**89.2**	81.3	74.6	**82.8**	**82.8**	81.2	**82.3**	81.7	81.7
$20, 10$	**68.2**	61.4	67.9	**68.2**	67.7	60.8	67.3	**69.0**	**72.1**	59.5	69.0	**72.1**
$20, 15$	74.7	71.6	**81.6**	**81.6**	82.9	71.4	**84.5**	**84.5**	**89.8**	79.4	89.4	**89.8**

Similar to the previous experiment, we also observe that the optimized mask ($\mathcal{V}$) achieves the best GSS on 17 over 18 settings, 10 of which are clear improvements over the three baseline choices for v. Overall, our experiments suggest that it is beneficial to optimize v, and that our framework is more successful in finding sketches with high GSS than other sketch construction methods.

Finally, Table 3.4 reports the best performing masks found by our optimization routine in each scenario. We observe that there is no fixed mask that consistently performs the best across all experiments. In addition, the maximum pruning depth observed is 3 (e.g., the algorithm terminates after 3 iterations of the outer loop because no possible GSS improvement can be found), which implies that dense masks are generally better for our benchmark sequences. In contrast, the best performing masks reported by Dutta et al. [22] are significantly sparser, such as $v = \{3, 9\}$ and $v = \{6\}$ for $k = 15$. We remark that this does not contradict our findings, as

Table 3.3: Comparing GSS (normalized to the scale of 0–100) of different masked minimizers using 3 different discrete construction methods and 6 settings of (w, k) on CHRXC. The best GSS observed for each combination of (w, k) and construction method is given in **bold**. The best GSS for each combination of (w, k) is further underlined.

	MINICEPTION UHS				PASHA UHS				Random Ordering			
(w, k)	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$
$10, 10$	57.4	0.0	55.1	**58.2**	**62.9**	43.8	60.1	**62.9**	26.5	21.0	28.7	**28.8**
$10, 15$	47.3	25.1	48.7	**50.1**	75.6	19.2	**76.9**	**76.9**	22.2	7.9	26.2	**26.5**
$15, 10$	**55.7**	0.0	57.6	**55.7**	51.0	44.4	**58.9**	**58.9**	27.1	**27.6**	25.3	27.1
$15, 15$	43.5	36.9	47.1	**50.9**	52.1	30.2	55.8	**63.2**	**17.2**	11.9	14.0	**17.2**
$20, 10$	**60.4**	0.0	48.9	**60.4**	43.5	30.7	55.1	**55.3**	18.9	20.2	23.9	**24.7**
$20, 15$	39.2	0.0	43.5	**47.6**	32.9	31.5	39.0	**39.9**	**13.2**	9.5	12.8	**13.2**

it was obtained on random sequences and Dutta et al. [22] compares parameterized syncmers by the root mean squared gap lengths metric.

The complete mask is a good initialization

We visualize the distribution of GSS across different masks. Fig. 3.16 (left) shows the scatter plot of all $2^w - 1$ masked minimizer schemes trained on BTR4 using $\mathcal{L}_{gss}$ with $w = 10$ and $k = 15$, grouped by the cardinality of v. Fig. 3.17 further shows the scatter plots of all $2^w - 1$ masked minimizers trained on BTR1, BTR2 and BTR3 using $\mathcal{L}_{gss}$ with $w = 10$ and $k = 15$, grouped by $|v|$. Across all experiments, we observe that the average GSS generally increases with $|v|$ in all experiments, which implies that the minimizer mask is a good default choice.

Table 3.4: The optimized masks found by our algorithm with 3 different training losses across 6 settings of (w, k) on CHRXC, denoted in the format $v_m \setminus v_{\mathrm{p}}$, where v_m is the complete minimizer mask, and v_{p} contains the pruned offsets.

(w, k)	Conservation Loss ($\mathcal{L}_{con}$)	Density Loss ($\mathcal{L}_{DM}$)	Combined Loss ($\mathcal{L}_{gss}$)
$10, 10$	v_m	$v_m \setminus \{7\}$	v_m
$10, 15$	v_m	$v_m \setminus \{5\}$	$v_m \setminus \{1, 8\}$
$15, 10$	$v_m \setminus \{6\}$	$v_m \setminus \{5, 8\}$	$v_m \setminus \{14\}$
$15, 15$	$v_m \setminus \{0, 2, 5\}$	$v_m \setminus \{7\}$	$v_m \setminus \{3, 7\}$
$20, 10$	v_m	$v_m \setminus \{9\}$	v_m
$20, 15$	$v_m \setminus \{10\}$	$v_m \setminus \{10\}$	v_m

Repeated sampling in homopolymer-rich sequences

One advantage of open-syncmers with $t > 1$ is the ability to avoid repeated sampling of identical k-mers in homopolymer substrings (i.e., substrings with repeated submer patterns) [24]. To confirm this, Fig. 3.16 (right) plots the GSS of all syncmer masks (with offsets in $[0, w - 1]$) and their complement masks on a synthetic sequence with $L = 100000$ and 0.2% homopolymer content. The dotted line shows the GSS of the minimizer mask, which expectedly performs worse than most open-syncmer masks (except for $v = \{0\}$) due to the repeated sampling pitfall.

In particular, because of the left-most tie breaking rule, every scheme whose mask contains the offset 0 (e.g., the minimizer mask, the open-syncmer mask with $v = \{0\}$, and all complement masks except the one where $0 \notin v$) suffers from high density. In contrast, we observe that the complement scheme with $v = [1, w - 1]$ achieves the best GSS (0.56). This is because it avoids the repeated sampling pitfall in the same way any open-syncmer scheme with $t > 0$ does, but otherwise performs like a minimizer scheme and does not suffer from the low coverage of syncmers.

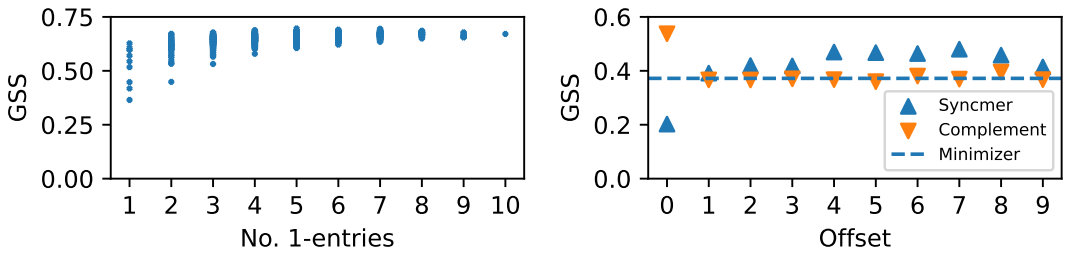


Figure 3.16: Left: GSS vs. number of 1-entries of all masked minimizers trained on the bacterial genome BTR4; Right: GSS vs. the offset position of various open-syncmer masks, and their complement schemes on a synthetic sequence with high homopolymer content.

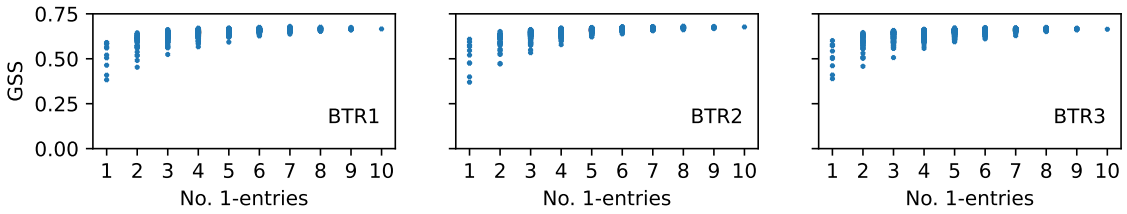


Figure 3.17: GSS vs. $|v|$ of all masked minimizers on bacterial genomes BTR1, BTR2 and BTR3.

Exploiting the relative density metric

This experiment demonstrates that without the coverage normalization step, the conservation-density ratio (i.e., relative conservation) can be exploited. We show that this exploitative behavior can be obtained by optimizing the loss function $\mathcal{L}_{exploit} \triangleq \sum_{i=t}^{n} \Delta(\mathbf{f}'_t, \mathbf{f})$. This loss function differs from $\mathcal{L}_{con}$ by swapping the template $\mathbf{g}$ in each pairwise Δ-distance term with $\mathbf{f}$. The purpose of this substitution is to isolate any training signal for density (which is implicitly encoded in the template) and to directly prioritize minimizing relative conservation. As minimizers schemes must select one position per (w, k)-window by construction, they do not suffer from this exploit. We thus train only the open-syncmer scheme $\mathcal{O}$ on a random sequence with $L = 1000$, using $\mathcal{L}_{exploit}$ with $w = 10$ and $k = 15$.

We plot the relative conservation (left-most column of Fig. 3.18) and coverage metrics (middle column of Fig. 3.18) obtained over 1000 epochs with $n \in \{1, 5, 10, 20\}$ sampled mutations per training epoch and offset $t \in \{6, 7, 8, 9\}$ (e.g., the corresponding masks are $v =$

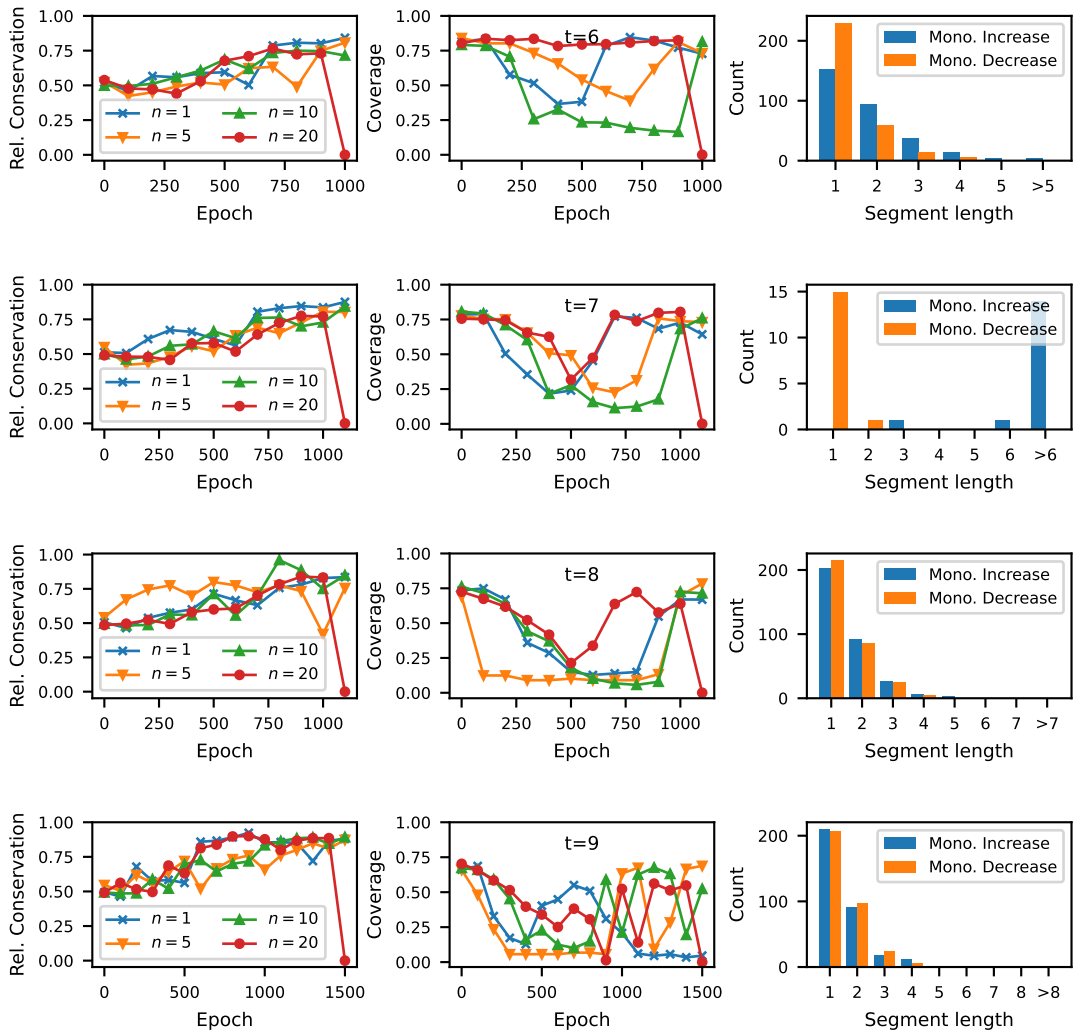


Figure 3.18: Finding the relative conservation exploit for various open syncmers using $\mathcal{L}_{exploit}$, $w = 10$ and $k = 15$ with (from top to bottom) offset $t \in \{6, 7, 8, 9\}$.

$\{6\}, \{7\}, \{8\}, \{9\}$). Generally, $\mathcal{L}_{exploit}$ improves relative conservation as expected. However, when $n = 20$, the optimizer finds the exploit mentioned in Section 3.4.3 after $1000 - 1500$ epochs, which causes both metrics to become 0. The resulting sketch consequently selects no k-mers (i.e., 0 coverage) and is trivially conserved when mutations are introduced (i.e., infinite conservation, which is manually set to 0 in the above plots).

We further plot the number of segments with monotonically increasing or decreasing priority scores at each segment length (right most column). For every value of t, the exploitative solution

contains no segment with more than $t-1$ consecutively decreasing scores. We note that the total count for $t = 7$ is significantly lower than other values of t because the solution contains several segments of monotonically increasing scores that are relatively long, which count towards the > 6 bucket. This result suggests that all lowest scoring k-mers are likely found within the first $t-1$ positions of their respective windows, and none are sub-sampled into the masked minimizer sketch.

3.5 Conclusion

This chapter studies the sketch design problem for various string sketching methods. The first part of this chapter introduces a novel framework called DEEPMINIMIZER for learning low-density *sequence-specific* minimizers. This is achieved via casting minimizer selection as optimizing a k-mer scoring function, parameterized by a deep neural network called PRIORITYNET. We further introduce a continuous relaxation of the density minimization objective via combining the PRIORITYNET with a complementary neural network, called TEMPLATENET. The TEMPLATENET pinpoints neighborhood of low-density score assignments (although these score assignments may not yield desirable properties of a minimizer sketch), and guides the PRIORITYNET to the neighborhood of low-density assignments around them. Coupling these networks leads to a fully differentiable proxy objective that can effectively be optimized through gradient-based learning techniques. The DEEPMINIMIZER method obtains better performance than previous state-of-the-art sequence-agnostic and sequence-aware minimizer selection schemes, especially on known hard tasks such as sketching the repetitive centromere region of Chromosome X.

The second part of this chapter studies a more general version of the sketch design problem, in which we extend the minimizer concept to masked minimizers and construct an optimization objective that simultaneously accounts various sketching metrics. We call this new metric the generalized sketch score (GSS) and develop a bi-level optimization framework to design masked

minimizers with high GSS for a specific reference sequence. We show that our algorithm finds combinations of masks and k-mer orderings that induce masked minimizer schemes with better GSS than other sketch construction methods. We additionally introduce a special category of complement masks that combine desirable properties of minimizers and syncmers. We demonstrate the robustness of these masks in both the standard setting and sketching a homopolymer-rich sequence that is known to be a pitfall for the minimizer method.

Chapter 4

Federated Neural Architecture Search

4.1 Introduction

Neural network architecture is the core element of deep learning's success on many perceptual tasks such as computer vision [58] and natural language processing [98]. Even though most renowned architectures in deep learning literature are hand-designed by domain experts, recent studies have suggested that searching for an optimal design that composes well-known building blocks can significantly improve performances in many task domains [44, 104, 113].

Formally, a neural network architecture can be written as a hierarchical feature extractor which explicitly takes the form of a directed acyclic graph $G \triangleq (V, E)$, where V and E respectively denote the sets of nodes and edges in G. Each node $v \in V$ denotes an intermediate feature representation z_v, which has been arbitrarily transformed from some input $\mathbf{x} \in \mathbb{R}^d$. We often associate $\mathbf{x}$ and the output of the network with a source node v_s and a sink node v_t, respectively. Each edge $e \equiv (v, v') \in E$ encodes an operator o_e that transforms z_v and forwards the result to v' to be aggregated as $z_{v'}$. The computation of any intermediate representation in G can then be recursively defined as:

$$z_{v'} \triangleq \sum_{(v,v') \in E} o_{(v,v')}(z_v) \ , \tag{4.1}$$

where we have assumed a simple additive aggregation rule for simplicity.

In a typical neural architecture search task, there are two design choices to be made: (1) the flow of information determined by the topology of G; and (2) the corresponding transformations of data features encoded by the edge operators o_e for every $e \in E$. To ensure feasibility, the space of edge operators is often restricted to a small, finite set $\mathcal{O}$ of well-known layers, such as linear transformation, convolution and pooling, thus reducing the problem to finding an optimal mapping $m : E \to \mathcal{O}$ that assigns an operator in $\mathcal{O}$ for every edge. Similar to the setting of the kernel selection problem, we are also interested in only representing the discrete structures and categorical types of these layers in $\mathcal{O}$. This does not include continuous and differentiable parameters (e.g., layer weights), for which optimization is well-studied.

Manually configuring the structure of G can be time-consuming and may not fully exploit the vast design space of possible architectures. To this end, neural architecture search (NAS) surfaces as a study that aims at designing practical search spaces for G, and developing corresponding search algorithms that efficiently explore high-performing network structures within these spaces. This revolutionary paradigm shift has led to the development of more efficient and effective deep learning models [80, 113].

Various strategies to formulate the NAS search space have previously been explored, such as composing complex architectures from modular components [80, 113], or continually evolving the network structures [75]. This thesis chapter focuses on the over-parameterized graph search space, whose goal is to distil high-performing a candidate network $G = (V, E)$ from a larger graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, such that $V \subseteq \mathcal{V}$, $E \subseteq \mathcal{E}$, and the edges in G facilitates information flow from a source node v_s to a sink node v_t.

Learning to distill high-performing structures, however, requires a sufficiently large amount of training data and compute resource to achieve. In the multi-party NAS setting, in which a number of clients wish to configure neural architectures to solve various correlated tasks given their own limited and private data, this core challenge consequently gives rise to a perplexing

dilemma. That is, individually performing local NAS in a per-client manner is potentially inaccurate and ineffective due to the lack of data, whereas a centralized approach that pools data to a global server would surely compromise the privacy of clients. To this end, the Federated Neural Architecture Search (FNAS) paradigm, which combines the privacy-preserving principles of federated learning [72] with traditional NAS, has recently emerged as a solution.

Nonetheless, existing FNAS solutions [44, 104] only focus on the scenario where client tasks are assumed to be identical, which is described in Definition 11. As such, these methods are restricted to finding a homogeneous solution across all clients. In practice, tasks will likely diverge and require customization of solution to achieve satisfactory performance. This chapter hence turns the attention to the *multi-task* FNAS problem, which is described in Definition 12. In particular, we introduce a novel federated personalized NAS framework (FEDPNAS) that is capable of customizing the model architecture for each task in the Federated learning (FL) workflow. The FEDPNAS method represents the model architecture for each task as a sub-network sampled from a large, over-parameterized network. The sampling distribution is subsequently learned together with the weights of the sampled network. Our contributions include:

- A novel architecture search space that modularizes each candidate network into a base component (shared across tasks) and a personalizable component. These components respectively capture the task-agnostic and task-specific information inferred from training data.
- A context-aware sampling distribution conditioned on specific task instance, which captures task-specific information and naturally incorporates personalization into architecture search.
- An new FNAS algorithm that optimizes for a common architecture, and subsequently fine-tunes this architecture to fit each local dataset. We further adopt a meta-learning objective to ensure that the common architecture distribution converges at a *vantage* point that is relevant and beneficial to all clients.

- A theoretical perspective on the role of our FL objective and thorough empirical analysis showing significant performance gain compared to the state-of-the-art FL and NAS methods.

This work is co-authored by Carl Kingsford, and was accepted to the Workshop on New Frontiers in Federated Learning (NFFL) at the Neural Information Processing Systems (NeurIPS) conference in 2021 [37].

4.2 Problem setting

The neural architecture search problem on an over-parameterized search space can be described as follows:

Definition 11 (Neural Architecture Search) *A neural architecture search instance is described by $\tau \triangleq \{\mathcal{G}, \mathcal{O}, \mathcal{D}\}$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denotes an over-parameterized graph, $\mathcal{O}$ is the set of all edge operators, and $\mathcal{D}$ denotes the observed task data. Further let $\mathcal{M}$ be the set of all functions $m : \mathcal{E} \rightarrow \mathcal{O}$ that assign a feature mapper to any edge in $\mathcal{G}$. Finally, let $F_\tau : \mathcal{M} \rightarrow \mathbb{R}$ be the performance measuring function of this task, which (1) executes a well-defined algorithmic procedure to optimize the parameters of the selected operators (e.g., applying gradient descent update with respect to the cross-entropy loss until convergence) and then (2) returns the predictive accuracy evaluated on the test set. The NAS objective is given by:*

$$m_* \quad = \quad \underset{m \in \mathcal{M}}{\operatorname{argmax}} \, F_\tau(m) \, , \tag{4.2}$$

where m_* denotes the optimally performing assignment function with respect to τ.

The problem statement above is specifically written for a single-task scenario, for which only one instance of τ requires optimal configuration. In this thesis, we further consider a new setting for neural architecture search, which focuses on concurrently configuring a set of n related tasks $\boldsymbol{\tau} \triangleq \{\tau_1, \tau_2 \ldots \tau_n\}$. We state the problem as follows:

Definition 12 (Multi-task Neural Architecture Search) *Let $\boldsymbol{\tau} \triangleq \{\tau_1, \tau_2 \dots \tau_n\}$ be a set of neural architecture search tasks such that every task $\tau_i \triangleq \{\mathcal{G}, \mathcal{O}, \mathcal{D}_i\}$ shares the same search space $\mathcal{G}$ and the operator set $\mathcal{O}$, but differs from one another by the given task data $\mathcal{D}_i$. Using the same notation of $\mathcal{M}$ in Definition 11 and let $\mathbf{m} = \{m_1, m_2 \dots m_n\}$ be a collection of mappers in $\mathcal{M}$, where each m_i corresponds to a candidate architecture for task τ_i. The performance measuring function for the multi-task NAS problem is then given by averaging the per-task performance:*

$$F_{\boldsymbol{\tau}}(\mathbf{m}) \quad \triangleq \quad \frac{1}{n} \sum_{i=1}^{n} F_{\tau_i}(m_i) \; . \tag{4.3}$$

Finally, we state the multi-task NAS objective as the following optimization task:

$$\mathbf{m}_* \quad = \quad \underset{\mathbf{m} \subset \mathcal{M}}{\operatorname{argmax}} \; F_{\boldsymbol{\tau}}(\mathbf{m}) \; . \tag{4.4}$$

4.3 Related work

The NAS literature can generally be classified into two separate lines of research: (1) designing search space, and (2) developing search strategy. As briefly mentioned above, this thesis will adopt the over-parameterized search space proposed by Bender et al. [8] and focus on developing new search strategies under this paradigm. This section provides a brief summary of various NAS methods in this direction (which we will simply refer to as NAS from this point onward) and some preliminary works in the multi-task NAS domain.

4.3.1 Cell-based over-parameterized search space

Most NAS methods converge on a cell-based organization of the over-parameterized network search space that was inspired by Pham et al. [80]. This entails factoring the search space graph $\mathcal{G}$ into a linear chain of modular cell blocks, each serves as an over-parameterized network by itself. Every cell has its own source and sink node, which respectively receives the output of

its previous cell and forwards its own output to the next cell. The NAS outcome is then the combined edge selection for all cells in $\mathcal{G}$, which is then achieved by specific search strategies.

4.3.2 Scheduled dropout

One major challenge of NAS is the expensive cost of evaluating candidate architectures, which involves sufficient training of the selected layer weights. Bender et al. [8] proposes to isolate this bottleneck from the performance measuring function through optimizing the entire over-parameterized network $\mathcal{G}$ once before conducting edge selection, thus avoiding the repeated training cost. The cost and the stability of optimizing such a giant model are kept manageable via a steadily increasing dropout rate (i.e., the probability of zeroing out the output of a neuron) as the training progresses. Bender et al. [8] conducts random search to find the best pruning of the trained $\mathcal{G}$, but this step can easily be extended with other black-box optimization methods.

4.3.3 Continuous relaxation

A main cause for inefficiency in NAS is often due to the intractability of the edge selection problem, which can be written as a high-dimensional integer programming objective. Explicitly, we cast the output domain of the mapping function m as the space of one-hot vectors with dimension $|\mathcal{O}|$ and express the computation at edge $e = (v, v')$ as:

$$z_{v'} \;=\; \sum_{i=1}^{|\mathcal{O}|} m_i(e) \cdot o_i(z_v) \;, \tag{4.5}$$

where o_i denotes the i^{th} operator in $\mathcal{O}$, $m_i(e) \in \{0, 1\}$ denotes the i^{th} entry of the one-hot vector $m(e)$, i.e., $\sum_{i=1}^{|\mathcal{O}|} m_i(e) = 1$. Liu et al. [64] then proposes the to alleviate this problem via approximating $m(e)$ with a continuous vector $\bar{m}(e) \in \mathbb{R}^{|\mathcal{O}|}$, called the *operator mixing weights* of e. We can then rewrite the above computation using a softmax operator:

$$z_{v'} \;\simeq\; \sum_{i=1}^{|\mathcal{O}|} \frac{\exp(\bar{m}_i(e))}{\sum_{j=1}^{|\mathcal{O}|} \exp(\bar{m}_j(e))} \cdot o_i(z_v) \;. \tag{4.6}$$

In this manner, the computation induced by any candidate architecture can be expressed entirely with continuous transformations that are fully differentiable with respect to all $\{\bar{m}(e)\}_{e\in\mathcal{E}}$ and the corresponding layer weights, hence reducing the NAS problem to a gradient-based optimization task. Xie et al. [104] subsequently proposes a similar approach that instead replaces the softmax operator in Eq. (4.6) with a Gumbel-softmax operator [49], thus obtaining candidate architectures with mixing weights that converge to samples of a categorical distribution (i.e., when its temperature parameter is annealed to 0), which better approximates Eq. (4.5). Hu et al. [44] further exploits this set up to perform differentiable edge sampling using the straight-through trick [49], thus avoids the expensive cost of training all layer weights at once.

4.3.4 Federated NAS

More recently, the NAS problem is also considered in the federated learning context [72]. This line of research studies a variant of the multi-task NAS problem introduced above, where the task data $\mathcal{D}_i$ are assumed to be privately hosted and cannot leave their respective silos. To address this problem, He et al. [35] adopts the differentiable framework by Liu et al. [64] to perform local NAS for each task τ_i and periodically synchronizes these local architectures via averaging both their operator mixing weights and layer weights. Nonetheless, we note that this aggregation scheme will mandate all local nodes to converge at a single architecture, which is not necessarily optimal when their respective tasks are heterogeneous. We will therefore focus on solving exactly the multi-task NAS objective given in Definition 12, which instead seeks to efficiently optimize an entire collection of architectures to optimally address each task.

4.4 Methodology

Our method, FEDPNAS, adopts the same continuous relaxation of the over-parameterized architecture space as suggested by Hu et al. [44]. Unlike [44], which lacks the ability to customize

architectures for individual tasks, our method achieves architecture personalization via factorizing the architecture space into: (1) a base component that is shared among all client models; and (2) a task-specific component, which personalizes across clients to best solve their respective tasks. We learn the operator mixing weights $\bar{m}(e)$ for all edges, which induce the joint distribution over all edge operators over two phases.

In the *federated* phase of FEDPNAS, a common overall architecture is learned such that all task-specific architectures can be quickly and optimally derived from it with minimal fine-tuning efforts. This learning objective is achieved by extending the MAML meta learning algorithm proposed by Finn et al. [28]. (Section 4.4.2). Subsequently, in the *adaptation* phase of FEDPNAS, each client separately performs local update of its task-specific component using private data. Unlike other meta learning objectives [28], which only focus on adapting the weights of a single architecture, this adaptation step will result in diverging personalized architectures that are optimal for their respective tasks. The remainder of this section describes these contributions in details.

4.4.1 Cell-based Architecture Space

Similar to Hu et al. [44], we express the over-parameterized architecture $\mathcal{G}_{\mathcal{O}}$ as a stack of modular cells, i.e., $\mathcal{G}_0 = \bigcup_{t=1}^{C} \mathcal{G}_t$, where C is the total number of cells and each cell $\mathcal{G}_t$ denotes a compact space of architecture constructed from the operator list $\mathcal{O}$. To obtain an architecture from $\mathcal{G}_{\mathcal{O}}$, we first distill an architecture module $G_t = (V_t, E_t)$ from each cell by selecting a subset of edges in $\mathcal{G}_t$. The propagation path of the input vector across these modules is then heuristically chosen and specified by a cell-level DAG. For example, a simple linear propagation scheme can be written as $G(\mathbf{x}) = G_C \circ G_{C-1} \cdots \circ G_1(\mathbf{x})$, where $\mathbf{x}$ denotes an input vector and each G_i is treated as a feature map.

The inner computation of each distilled module G_t is similarly defined as in Section 4. Given an input vector $\mathbf{x}$, we then sample an operator o_e on each edge $e \equiv (v, v') \in E_t$ from the vocab-

ulary $\mathcal{O} \triangleq [o_1, o_2, \ldots, o_D]$ using a parameterized categorical distribution p_e^t. Hu et al. [44] assumes this edge sampling distribution is independent of the input $\mathbf{x}$ and any intermediate feature mapping of $\mathbf{x}$, hence does not make use of these context information in deciding the distilled architecture. In the context of *horizontal* NAS, we instead argue that these information are valuable in guiding client-level architecture adaptations and consequently parameterize p_e^t as conditional sampling distributions, which jointly model a hierarchical decision process (Section 4.4.3).

The feature aggregation scheme for any arbitrary node $v' \in V_t$ is then given as:

$$z_{v'} \;=\; \sum_{e \equiv (v,v')}^{e \in E_t} \mathbb{E}_{r \sim p_e^t} \left[r^\top \xi(z_v) \right] \;\simeq\; \sum_{e \equiv (v,v')}^{e \in E_t} \sum_{i=1}^{s} r_{ei}^\top \, \xi(z_v) \;, \tag{4.7}$$

where each r_{ei} is a one-hot vector drawn from p_e^t and $\xi(z_v) \triangleq [o_1(z_v), o_2(z_v), \ldots, o_D(z_v)]$ is the concatenation of all possible transformations of z_v using the operators in $\mathcal{O}$. Even though the computation above requires sampling from a categorical distribution, it can be made differentiable with respect to the parameters of p_e^t using the straight through Gumbel-softmax trick [49].

Unlike the original design, which assumes similar importance for every cell in the architecture stack, we opt to split our cell-based architecture into two component stacks with different roles to facilitate our personalized architecture search goal. Specifically, our search space contains: (a) a *base stack* $\mathcal{G}_b = \{\mathcal{G}_b^1, \mathcal{G}_b^2, \ldots, \mathcal{G}_b^{C_b}\}$, which aims to capture a feature map that is universally useful to all tasks; and (b) a *personalized stack* $\mathcal{G}_p = \{\mathcal{G}_p^1, \mathcal{G}_p^2, \ldots, \mathcal{G}_p^{C_p}\}$, which will be adapted using client data to capture task-specific features. Fig. 4.1 summarizes the relationship of these components.

Generally, we assume that the tasks are broadly related and diverge in finer details. Therefore, the base architecture stack is designed to be significantly more expressive than the personalized stack, via employing a larger operator vocabulary and a more sophisticated propagation scheme as shown in Fig. 4.1. In addition, as we will subsequently discuss in Section 4.4.2, our objective is formulated such that its gradient evaluation will require approximating the Hessian of the personalized component. As such, a minimally expressive personalized stack is also a practical design to lower the computational cost.

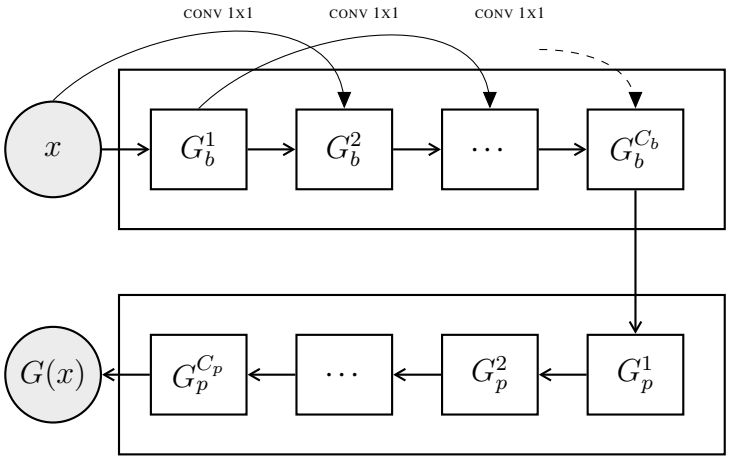


Figure 4.1: Feature mapping induced by the component stacks of our architecture space. Each cell in the base stack receives outputs from two previous cells, whereas each cell in the personalized stack receives outputs from only one previous cell.

4.4.2 Personalized Federated Learning Objective

In this section, we will now discuss the learning objective of FEDPNAS, which enables the discovery of personalized architecture. Let $\theta = \{\theta_b, \theta_p\}$ respectively denote all trainable parameters of the two component stacks above. In particular, $\theta_b = \{W_b, \Pi_b\}$ contains the concatenated weights W_b of all edge operators; and the concatenated parameters $\Pi_b = [\Pi_e^t]_{t\in[C_b],e\in E_t}$ of the joint edge sampling distribution. Likewise, $\theta_p = \{W_p, \Pi_p\}$ contains their counterparts in the personalized stack. Given N local clients with tasks $\{\Omega_1, \Omega_2, \ldots, \Omega_N\}$, the straight-forward extension of FL objective to the NAS problem defined by this search space can be written as:

$$\theta_* \quad = \quad \underset{\theta}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} F_{\Omega_i}(\theta) \; . \tag{4.8}$$

McMahan et al. [72] proposes a privacy preserving approach to optimize this objective by alternating between two communication steps of the model parameters. At any iteration $t \geq 0$:

- Each local client performs a local gradient step with its current parameter θ_i^t and sends the suggested update $\bar{\theta}_i^t = \theta_i^t + \lambda \nabla_\theta F_{\Omega_i}(\theta_i^t)$ to a central server.
- The central server then computes $\theta_{\text{SERVER}}^t = \frac{1}{N}\sum_{i=1}^{N} \bar{\theta}_i^t$ and broadcasts the aggregated pa-

rameters θ^t_{SERVER} to all clients.

- Each local client performs the update $\theta_i^{t+1} \leftarrow \theta^t_{\text{SERVER}}$ and prepares for the $(t+1)^{\text{th}}$ communication round.

This FL scheme implies that all clients will follow the same architecture distribution after the last communication round, which is not necessarily optimal in a heterogeneous task setting. To address this, our framework instead adopts the MAML meta learning objective [28], which aims to find a favorable initialization of θ that yields maximum averaged performance *given an expected adaptation step*. This is different from Eq. 4.8, which does not directly optimize for this initialization, but approximates it using the instantaneous averaged performance. Explicitly, this is achieved by the following objective:

$$\theta_* \quad = \quad \arg\max_{\theta_b,\theta_p} \frac{1}{N} \sum_{i=1}^{N} F_{\Omega_i} \left(\theta_b, \theta_p + \lambda \nabla_{\theta_p} F_{\Omega_i}(\theta_b, \theta_p)\right) \ , \tag{4.9}$$

in which the difference from Eq. (4.8), highlighted in red, models a gradient ascent update with step size λ to the aggregated personalized parameters θ_p, which is to be conducted by each client at the end of the federated phase. Intuitively, this gradient step acts as a regularization term which favors θ that are simultaneously close to all task-specific optima and yield the best averaged performance after the adaptation phase.

We now derive the gradient of this objective and discuss a practical algorithm to perform its computation. First, we let $\tilde{\theta}^t \triangleq \left(\theta_b^t, \theta_p^t + \lambda \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t)\right)$ denote the anticipated update of θ at time t of some arbitrary client. Then, dropping the client index for clarity, we subsequently derive the gradient ascent update for each client pertaining to the above personalized FL objective:

$$\begin{aligned}
\bar{\theta}^t &= \theta^t + \lambda \nabla_{\theta^t} F_\Omega \left(\tilde{\theta}^t\right) \\
&= \theta^t + \left(\lambda \nabla_{\theta^t} \tilde{\theta}^t\right) \left(\nabla_{\tilde{\theta}^t} F_\Omega(\tilde{\theta}_t)\right) \\
&= \theta^t + \begin{bmatrix} \lambda \mathbf{I} & \lambda^2 \nabla_{\theta_b^t} \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \\ \mathbf{0} & \lambda^2 \nabla^2_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \end{bmatrix} \left(\nabla_{\tilde{\theta}^t} F_\Omega(\tilde{\theta}^t)\right) \ ,
\end{aligned} \tag{4.10}$$

where we applied chain rule in the second equality and subsequently expanded $\lambda \nabla_{\theta^t} \tilde{\theta}^t$ in the

third equality. The second-order gradient terms $\nabla_{\theta_b^t}\nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)$ and $\nabla^2_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)$, however, are expensive to evaluate exactly. In order to derive practical computations of these terms, we note that the first-order Taylor approximation of the Hessian $\nabla^2_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)$ can be written as:

$$\begin{aligned}
\nabla^2_{\theta^t}F_\Omega(\theta_b^t,\theta_p^t) &= \begin{bmatrix} \nabla^2_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t) & \nabla_{\theta_b^t}\nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \\ \nabla_{\theta_p^t}\nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t) & \nabla^2_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \end{bmatrix} \\
&\simeq \begin{bmatrix} \nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t) \\ \nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \end{bmatrix} \begin{bmatrix} \nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t) \\ \nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \end{bmatrix}^\top \\
&= \begin{bmatrix} \nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t) & \nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \\ \nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t) & \nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \end{bmatrix} . \qquad (4.11)
\end{aligned}$$

Matching appropriate terms in the above derivation then implies the following approximations, which can be computed with a single forward-backward pass of the architecture:

$$\nabla_{\theta_b^t}\nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \simeq \nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) , \qquad (4.12)$$

$$\nabla^2_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \simeq \nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) . \qquad (4.13)$$

Plugging this back to Eq. (4.10) gives:

$$\bar{\theta}^t \simeq \theta^t + \begin{bmatrix} \lambda\mathbf{I} & \lambda^2\nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \\ \mathbf{0} & \lambda^2\nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t) \end{bmatrix} \left(\nabla_{\tilde{\theta}^t}F_\Omega(\tilde{\theta}^t)\right) , \qquad (4.14)$$

where the term $\nabla_{\tilde{\theta}^t}F_\Omega(\tilde{\theta}^t)$ requires another forward-backward pass to compute (i.e., $\tilde{\theta}^t$ is computed in the same pass with the two approximated gradient terms above). Overall, this results in a local update scheme which uses two forward-backward passes of the architecture per iteration:

- Compute $F_\Omega(\theta^t)$ in the first forward pass.
- Perform backpropagation to obtain $\nabla_{\theta^t}F_\Omega(\theta^t) = \Big[\nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t), \nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)\Big]$.
- Compute $\nabla_{\theta_b^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)$ and $\nabla_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)$.
- Compute $\tilde{\theta}^t = \Big(\theta_b^t, \theta_p^t + \lambda\nabla^\top_{\theta_p^t}F_\Omega(\theta_b^t,\theta_p^t)\Big)$.

- Compute $F_\Omega(\tilde{\theta}^t)$ in the second forward pass.
- Perform backpropagation to obtain $\nabla_{\tilde{\theta}^t} F_\Omega(\tilde{\theta}^t)$.
- Compute the update in Eq. (4.14).

4.4.3 Context-Aware Operator Sampling

Finally, this section describes the parameterization of the joint operator sampling distribution over the edges of $\mathcal{G}_O$. Let $O_G \triangleq \cup_{t\in[C]} O_{G_t}$ be the set of all selected edge operators in a distilled architecture G, where $O_{G_t} \triangleq \{o_e^t \in \mathcal{O}\}_{e\in E_t}$ in turn denotes the set of all selected edge operators in cell G_t (i.e., in a single stack architecture). Hu et al. [44] then assumes a fully factorizable sampling distribution:

$$p(O_G) \;=\; \prod_{t\in[C]} p(O_{G_t}) \;=\; \prod_{t\in[C]} \prod_{e\in E_t} p_e^t(o_e^t; \Pi_e^t)\;, \tag{4.15}$$

where Π_e^t denotes the learnable parameters of the edge sampling distribution p_e^t for every $t \in [C]$ and $e \in E_t$. This formulation, however, does not factor in the important context information carried by the input instance $\mathbf{x}$ and its subsequent embeddings as $\mathbf{x}$ propagates through G. While this approach might be sufficient when only one architecture needs to be distilled from $\mathcal{G}_\mathcal{O}$, it is challenging to extend to the *horizontal* NAS setting because every edge in the over-parameterized architecture $\mathcal{G}_\mathcal{O}$ is required to maintain and optimize its own set of categorical distribution parameters. In the context of our *horizontal* NAS problem, this means that the set of parameters that need to be personalized will also scale with the size of the architecture, hence posing both a computational and a convergence problem.

To overcome this issue, we will instead look at the conditional sampling distribution $p(O_g \mid \mathbf{x})$ and subsequently model architecture distillation as a Markov chain decision process, where each cell in the architecture stack will sequentially determine its edge operators given the input feature it receives from previous cells. For simplicity, we give our factorization of $p(O_G \mid \mathbf{x})$ below as if the architecture contains a single stack of cells and uses a linear propagation scheme,

but it would be trivial to extend this formulation to any other propagation scheme:

$$\begin{aligned} p(O_G \mid \mathbf{x}) &= p(O_{G_1} \mid \mathbf{x}) \prod_{t=2}^{C} p(O_{G_t} \mid O_{G_{t-1}}, \mathbf{x}) \\ &\simeq \prod_{t=1}^{C} \prod_{e \equiv (v,v')}^{e \in E_t} p_e^t \left(o_{v,v'}^t \mid z_v \right) \end{aligned} \tag{4.16}$$

where z_v denotes the feature vector at an arbitrary node v in G and in the factorization above, we have also assumed that z_v acts as the sufficient statistics of the random variable $o_{v,v'}^t$.

The advantage of this formulation is two-fold. First, it provides a natural mechanism to incorporate context information into the edge selection process, thus allowing FEDPNAS to efficiently memorize architecture distillation patterns across clients. Second, the above factorization of $p(O_G \mid \mathbf{x})$ as a product of edge-wise conditional probabilities further reveals a compact representation of all sampling parameters using a single neural network ψ, which reduces the number of parameters that require fine-tuning in the local adaptation phase. That is, for any edge $e \equiv (v, v') \in E_t$, its context-aware operator distribution conditioned on z_v is given as $p_e^t(o_{v,v'}^t \mid z_v) \triangleq \mathrm{Cat}\left(D, \psi\left(z_v\right)\right)$, where D denotes the number of operators in $\mathcal{O}$ and the neural network ψ maps the z_v to the event probabilities of a categorical distribution.

4.5 Empirical Study

We conduct experiments to showcase the performance of FEDPNAS compared to other NAS and FL benchmarks. Our empirical studies are conducted on two image recognition datasets: (a) the CIFAR-10 dataset [55] which aims to predict image labels from 10 classes given a train/test set of $50000/10000$ colour images of dimension 32×32 pixels; and (b) the MNIST dataset [57] which aims to predict handwritten digits (i.e. 0 to 9) given a train/test set of $60000/10000$ grayscale images of dimension 28×28 pixels. Our search space entails 2^{40} possible architectures. Table 4.1 gives the list of operators available for sampling at every edge of the over-parameterized architecture search space. The implementation details of each operator type follows that of SNAS

Table 4.1: List of operators to be sampled from at every edge and their respective number of parameters in the CIFAR-10 experiment (32 channels) and the MNIST experiment (16 channels). MASTEREDGE denotes the combination of all operators at every edge in the network and MASTERARCHITECTURE denotes the combination of all operators in the network (total of 32 edges).

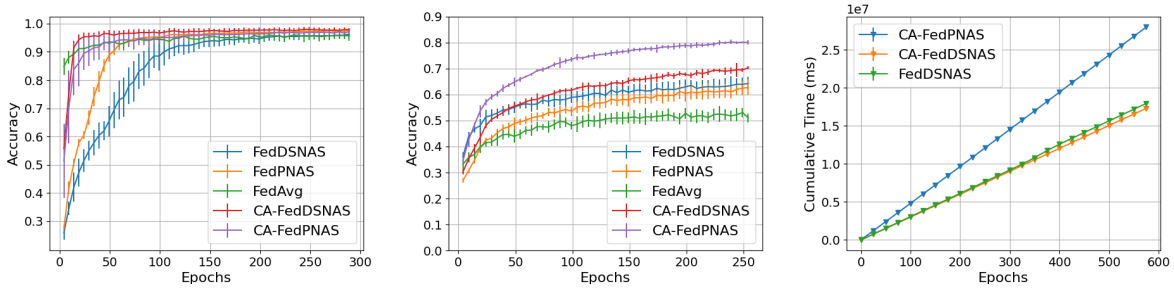
OPERATOR	FILTER SIZE	NO. PARAMS (CIFAR-10)	NO. PARAMS (MNIST)
AVGPOOL	3×3	64	32
MAXPOOL	3×3	64	32
SEPCONV	3×3	2752	864
SEPCONV	5×5	3776	1376
DILCONV	3×3	1376	432
DILCONV	5×5	1888	688
HOURGLASSCONV	$7 \times 1, 1 \times 7$	14400	3616
SKIPCONNECT	N/A	0	0
MASTEREDGE		29632	9184
MASTERARCHITECTURE		948224	293888

[104] and DSNAS [44]. Even though the master architecture has nearly a million parameters, we note that each learning batch only samples a fraction of these parameters to update.

We compare two variants of our framework, CA-FEDPNAS (with context-aware operation sampler) and FEDPNAS (without the operation sampler), against: (a) FEDAVERAGING of a fixed architecture to justify the need for NAS in FL; (b) FEDDSNAS - which trivially extends DSNAS to the FL setting (Eq. (4.8)); and (c) CA-FEDDSNAS, which extends FEDDSNAS with our context-aware sampler.

4.5.1 Heterogeneous predictive tasks

We first design a control experiment to test our framework on heterogeneous tasks and demonstrate the necessity of architecture personalization. To simulate this scenario, we first distribute the data i.i.d across clients (10000/2000 and 12000/2000 training/test images per client for CIFAR-10 and MNIST datasets respectively). Then, we independently apply a different transformation to each partitioned dataset. Input images within the same train/test set is subject to the same transformation. In both our experiments, the client datasets are subjected to rotations of $-30°, -15°, 0°, 15°$ and $30°$ respectively. Fig. 4.2 below shows the performance of all the methods in comparison, plotted against number of search epochs and averaged over the above rotated variants of CIFAR-10 and MNIST datasets.



(a) (b) (c)

Figure 4.2: Plotting average classification accuracy of various methods against no. training epochs on heterogeneous tasks derived from (a) MNIST and (b) CIFAR-10 dataset. Figure (c) compares cumulative running time of various methods against no. training epochs on the CIFAR-10 dataset.

On the MNIST dataset (Fig. 4.2b), all methods converge to a similar performance. Among the NAS benchmarks, FEDPNAS and FEDDSNAS both converge slower than FEDAVG and start off with worse performance in early iterations. This is expected since FEDAVG does not have to search for the architecture and it is likely that the default architecture is sufficient for the MNIST task. On the other hand, we observe that both CA-FEDPNAS and CA-FEDDSNAS converge

much faster than their counterparts without the context-aware operation sampler component. This shows that making use of contextual information helps to quickly locate regions of high-performing architectures, especially on similar inputs.

On the CIFAR-10 dataset (Fig. 4.2a), we instead observe significant gaps between the worst performing FEDAVG and other NAS methods. This is likely because the default architecture does not have sufficient learning capability, which confirms the need for customizing solutions. Among the NAS benchmarks, we again observe that both CA-FEDPNAS and CA-FEDDSNAS outperform their counterparts without our operation sampler, which confirms the intuition above. Most remarkably, our proposed framework CA-FEDPNAS achieves the best performance (0.8) and significantly outperformed both variants of federated DSNAS (0.71 for CA-FEDDSNAS and 0.63 for FEDDSNAS).

Lastly, Fig. 4.2c shows the runtime comparison between three methods on the CIFAR-10 experiment. In terms of sampling time, we observe that there is negligible overhead incurred by using our context-aware sampler (CA-FEDDSNAS vs. FEDDSNAS). The time incurred by our update (CA-FEDPNAS) scales by a constant factor compared to CA-FEDDSNAS since we use exactly one extra forward-backward pass per update.

4.5.2 Tasks with varying heterogeneity levels

We expand the above study by subsequently investigating the respective performances of CA-FEDPNAS and FEDDSNAS on tasks with varying levels of heterogeneity. At low levels of heterogeneity, we deploy these methods on 5 sets of slightly rotated MNIST images. At high levels of heterogeneity, we employ a more diverse set of transformations on MNIST images, such as hue jitter and large angle rotations of 90° and -90°. Table 4.2 shows the respective result of each task from these two settings. We observe that our method CA-FEDPNAS achieves better performance on most tasks and the performance gaps on tasks with higher heterogeneity are more pronounced (i.e., up to 7% improvement on ROTATE 90 task). This clearly shows the importance

Table 4.2: Predictive accuracy of CA-FEDPNAS FEDDSNAS on tasks with varying heterogeneity levels. ROTATE X denotes a rotation transformation of X° on client data; VANILLA denotes the original MNIST images; and HUEJITTER X denotes a hue jitter transformation of training images by a factor of X. The best performance in each row is in bold font.

HETEROGENEITY	TASK DESCRIPTION	FEDDSNAS	CA-FEDPNAS
LOW	ROTATE -30	0.947	**0.978**
	ROTATE -15	0.973	**0.976**
	VANILLA	**0.988**	0.985
	ROTATE 15	0.986	**0.987**
	ROTATE 30	0.972	**0.981**
HIGH	HUEJITTER -0.5	0.966	**0.978**
	HUEJITTER 0.5	0.967	**0.972**
	VANILLA	0.988	**0.989**
	ROTATE -90	0.892	**0.932**
	ROTATE 90	0.866	**0.932**

of architecture personalization when the training tasks are significantly different and justifies our research goal.

4.5.3 Knowledge transfer to completely new tasks

Finally, we conduct an ablation study to assess the quality of the *pre-adaptation* architecture distributions respectively discovered by CA-FEDPNAS and FEDDSNAS. In particular, we will leverage these learned distributions, which supposedly capture the broad commonalities of the task distribution, to generalize to completely unseen tasks (i.e., tasks that do not participate in the federated learning phase). To simulate this scenario, we train both methods on five clients

whose local data consist of 12000 rotated CIFAR-10 images (i.e., in the range of $\pm 30^{\circ}$), similar to the setting of the first experiment. During the evaluation phase, however, we supply each local client with 2000 test images subjected to related but completely unseen transformations (i.e., 90° and -90° rotations).

Table 4.3: Predictive accuracy (averaged over 5 clients) and standard deviation of CA-FEDPNAS and FEDDSNAS on two unseen tasks (CIFAR-10).

UNSEEN TASK DESCRIPTION	FEDDSNAS	CA-FEDPNAS	FEDDSNAS (RETRAINED)	CA-FEDPNAS (RETRAINED)
ROTATE -90	0.545 ± 0.04	0.578 ± 0.09	0.699 ± 0.12	$\mathbf{0.734 \pm 0.17}$
ROTATE 90	0.553 ± 0.12	0.569 ± 0.06	0.673 ± 0.13	$\mathbf{0.727 \pm 0.22}$

We summarize our results in Table 4.3 above. First, we measure the performance of CA-FEDPNAS and FEDDSNAS without any weight retraining. When received no additional information from the unseen tasks, both methods perform poorly as expected. While CA-FEDPNAS achieves better predictive accuracy, the performance gap in this scenario is negligible. To provide additional clues for adaptation, albeit minimal, we retrain the weights of each local model with 200 images that are rotated according to their respective unseen task description. With only 100 retraining iterations on limited data, CA-FEDPSNAS already outperforms FEDDSNAS (5% and 8% improvement respectively on two unseen tasks). This implies that CA-FEDPNAS has captured more accurately the broad similarity of the task spectrum and requires minimal additional information to successfully adapt to unseen tasks.

4.6 Conclusion

We demonstrate that federated learning for multi-task scenarios requires extensive personalization on the architecture level to obtain good predictive performance. This paper identifies two

potential sources of model personalization: (1) task-personalization, which aims to select architectures best suited for specific learning objectives; and (2) context-personalization, which aims to select architectures best suited for specific input samples. To incorporate these aspects of personalization into Federated NAS, we propose FEDPNAS which consists of two main components: (1) a context-aware operator sampler which learns a sampling distribution for feature maps along a master architecture; and (2) a personalized federated learning objective which anticipates client fine-tuning and guides the federated model update to regions that tolerate future local updates. Finally, we demonstrate the performance gain of our method compared to existing NAS and FL frameworks on several real-world benchmarks.

Orthogonal to this work, we have also investigated the related problem of black-box model fusion (BBMF). It differs from standard FL in two aspects. First, the client models are pre-trained and they are unable to coordinate their parameter update. Due to this constraint, the aim of model fusion is to aggregate these artifacts into a more accurate predictor, given only the ability to query their predictions at any arbitrary input. In addition, the BBMF problem demands a stronger level of privacy compliance, which avoids exposing both training data and internal architectures to other clients.

To this end, we present the first collective model fusion framework for multiple experts with heterogeneous black-box architectures. The proposed method addresses the key issue of how black-box experts interact to understand the predictive behaviors of one another, how these understandings can be represented and shared efficiently among themselves, and how the shared understandings can be combined to generate high-quality consensus prediction. The performance of the resulting framework is analyzed theoretically and demonstrated empirically on several datasets. This work is co-authored by Carl Kingsford, Nghia Hoang and Bryan Low, and was published at the International Conference on Machine Learning (ICML) 2019 [39].

Chapter 5

Meta-Learning for Heterogeneous Tasks

5.1 Introduction

Few-shot learning (FSL) is a challenging problem where the goal is to learn new concepts with only a small number of labeled samples, similar to how humans learn new things by incorporating prior knowledge and context. One promising approach to tackle FSL problems is meta-learning, which learns to extract and generalize transferable meta-knowledge from a distribution of tasks and quickly adapt it to unseen tasks.

Many meta-learning methods [1, 81, 107] are built upon the model-agnostic meta learning (MAML) framework [28]. The MAML method aims to learn a single set of model parameters (for any arbitrary network architecture) that can be quickly fine-tuned to deliver high performance on unseen tasks. However, most MAML variants are grounded in the context of a homogeneous task distribution where all tasks originate from the same concept domain and thus implicitly assume that all transferable knowledge are globally shared among all tasks [100]. This assumption, however, constrains the generalization capacity of these meta-learners when handling multi-modal task distributions, for which the task-specific optimal parameters could diverge significantly from one another. For example, if the task distribution consists of modes that are far apart (e.g., animal and vehicle recognition tasks), it would be impossible to find an

initialization that is simultaneously close to all modes.

Recent work has demonstrated that the generalization of MAML and, by extension, its many variants, is indeed related to the similarity of tasks in the training distribution [112]. This perspective aligns with many previous clustering approaches that aim to detect homogeneous clusters of tasks which MAML-based learners can be effectively applied [100, 105, 112]. Zhou et al. [112] seeks to learn an ensemble of initializations, each of which is set to represent a cluster of task (i.e., a mode in the task distribution). This is achieved via augmenting the MAML loss function with an assignment step. The cluster assignment heuristic, however, is conditioned on the single-mode, vanilla MAML initialization and thus is likely not optimal in a multi-modal setting.

Alternatively, Yao et al. [105] and Vuorio et al. [100] propose to implicitly cluster tasks using the embedding vectors of their few-shot data. In particular, Vuorio et al. [100] applies a *modulation network* on the learned task embedding to modulate the meta-initialization of the predictor model, yielding the task-specific parameters. Yao et al. [105] adopts a similar idea, but further imposes explicit hierarchical structure on the task space through jointly optimizing several task cluster centroids. The estimated parameter modulation is then applied to the nearest centroid based on their embedding distance.

While both methods are capable of addressing the task heterogeneity challenge, they suffer from significant increase in parameter complexity since their respective modulation networks must scale with the size of the predictor model (e.g., for an average convolutional architecture with millions of parameters, the modulation network is essentially a million-output map). Even when the modulation is applied layer-wise, learning to generate that many variables is still a challenging task. This thus prevents applying these tactics on larger architectures.

To overcome this weakness, we develop a new heterogeneous meta-learning strategy that efficiently captures the multi-modality of the task distribution via modulating the routing between neurons in the network, instead of directly modulating the network weights. Our approach is

inspired by the ShuffleNet architecture [108], which employs convolutional channel shuffling to encode a highly expressive solution space. The phenomenal success of ShuffleNet, which achieves comparable performance to state-of-the-art models that have many-fold more parameters, suggests that adapting the routing configuration (i.e., implied by the channel shuffling order) can potentially emulate the modulation of many neurons without incurring the extra computational costs.

This insight therefore motivates us to reformulate the weight modulation network in previous heterogeneous meta-learning work [100, 105] as a routing modulation network that controls task-specific shuffling of convolution channels. In particular, given a task embedding vector, our modulation network learns to generate a permutation matrix which simulates the channel shuffling operator when multiplied with the output of a convolution layer. To model this permutation network, one can adopt the Gumbel-Sinkhorn layer [73], which differentiably transforms general square matrices to discrete permutation matrices in the limit of a temperature parameter. The permutation network thus can be optimized via learning a mapping $f : \mathbb{R}^z \rightarrow \mathbb{R}^{C^2}$, where z and C are respectively the task embedding dimension and the number of convolutional channels.

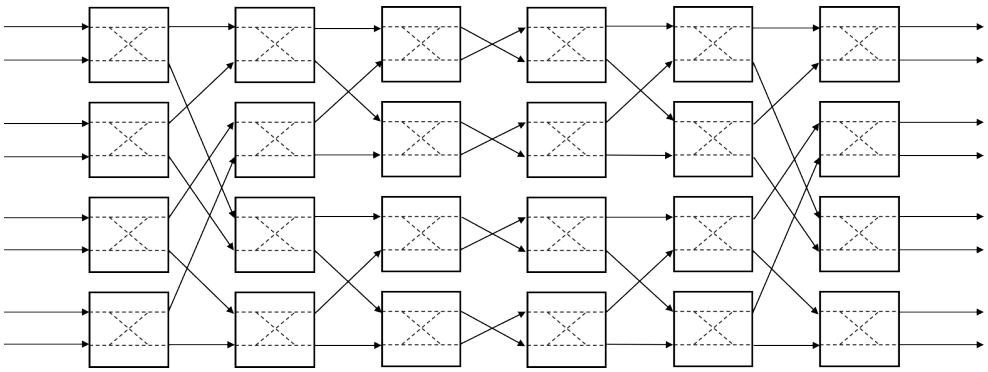


Figure 5.1: Beneš network for $C = 8$. Input channels are grouped pairwise. Each square block denotes a learnable binary switch which either maintains or permutes the ordering of its respective pairs. Routing between layers of switches are performed in a deterministic manner [9].

Nonetheless, accurately learning a dense $C \times C$ matrix given the few shot training data

can still be challenging, especially for large convolutional layers with many channels. To this end, we propose an even more compact formulation of the permutation module based on the Beneš routing network [9], which can emulate any C-permutation using at most $C \log_2 C$ binary switches that pairwise permute adjacent indices (Fig. 5.1). Finally, to enable end-to-end learning of this compact permutation network, we approximate the discrete switches by applying the same Gumbel-softmax transformation in [73] to 2×2 general matrices. In summary, the contributions of this chapter include:

- A more efficient heterogeneous meta-learning framework that estimates the different modalities in the task distribution via modulating the network routing configurations. This modulation operator takes the form of a permutation network that performs channel shuffling based on the few shot training data of a task. Our meta routing modulation (MRM) framework is presented in Section 5.4.
- A compact formulation of the above permutation network based on a continuous relaxation of the classical Beneš network [9], which we call the Gumbel-Beneš layer (Section 5.4). The Gumbel-Beneš layer trains efficiently with limited data and scales better in the number of network channels than the state-of-the-art Gumbel-Sinkhorn method.
- A demonstration of our approach on various multi-modal meta-learning benchmarks. We show that our framework outperforms existing methods in both generalization accuracy and runtime (Section 5.5).

This work is co-authored by Carl Kingsford, and is currently under review at the Neural Information Processing Systems (NeurIPS) conference, 2023.

5.2 Problem setting

In the meta-learning setting, we are given a task distribution $\mathcal{T}$, where each task $T_i \sim \mathcal{T}$ consists of a dataset $\mathcal{D}_i$ and a learning objective $\mathcal{L}_i$. Similar to many other meta-learning studies, we adopt

a simplified setting where all tasks in $\mathcal{T}$ share the same learning objective $\mathcal{L}$ and each dataset $\mathcal{D}_i = \{\mathbf{x}_{ij}, \mathbf{y}_{ij}\}_{j=1}^n$ contains n-shot supervised learning samples. The goal of meta-learning is to train a meta-model $\mathcal{M}_*$ that maps any task T_i to a parameter vector θ_i in the weight space of some predictor model G, such that $\mathcal{M}_*$ minimizes the expected learning loss (over random tasks):

$$\begin{aligned} \mathcal{M}_* &= \underset{\mathcal{M}}{\operatorname{argmin}}\, \mathbb{E}_{T_i \sim \mathcal{T}} \left[\frac{1}{n} \sum_{j=1}^{n} \mathcal{L}\left(G(\mathbf{x}_{ij}; \theta_i \triangleq \mathcal{M}(T_i)), \mathbf{y}_{ij} \right) \right] \\ &= \underset{\mathcal{M}}{\operatorname{argmin}}\, \mathbb{E}_{T_i \sim \mathcal{T}} \left[\mathcal{L}^\dagger \left(\mathcal{M}(T_i), \mathcal{D}_i \right) \right] , \end{aligned} \tag{5.1}$$

where $\mathcal{L}_G^\dagger(\theta, \mathcal{D})$ denotes the averaged objective value evaluated on model architecture G with parameters θ over all data points in $\mathcal{D}$. Towards this goal, the MAML framework [28] models $\mathcal{M}_*(T_i)$ as a fine-tuning gradient descent step with respect to $\mathcal{D}_i$ given some base initialization θ_*. That is, $\mathcal{M}_*(T_i) \triangleq \theta_* - \eta \nabla_\theta \mathcal{L}_G^\dagger(\theta_*, \mathcal{D}_i)$, where η denotes the step size. To obtain the base initialization θ_*, Finn et al. [28] proposes to optimize the following loss function:

$$\theta_* = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{T_i \sim \mathcal{T}} \left[\mathcal{L}_G^\dagger \left(\theta - \eta \nabla_\theta \mathcal{L}_G^\dagger \left(\theta, \mathcal{D}_i^t \right), \mathcal{D}_i^v \right) \right] , \tag{5.2}$$

where $\{\mathcal{D}_i^t, \mathcal{D}_i^v\}$ denotes the train-validation split of $\mathcal{D}_i$. Intuitively, the goal of this loss function is to find a single initialization θ_* such that given the fine-tuning step at the time of evaluating $\mathcal{M}_*(T_i)$, the adapted parameters will yield the best performance in expectation.

5.3 Related work

5.3.1 Meta-learning

Existing meta-learning approaches can be broadly classified into three families: metric-based, model-based, and optimization-based methods. Model-based approaches [34, 95] aim to recognize the task identity from its few-shot data and use the task identity to adjust the model state accordingly. While these methods perform well on certain task domains, they require fixing the model architecture and thus are difficult to apply on arbitrary use cases. Metric-based

methods [92, 99] learn a task similarity metric (based on observed data) which can be used to perform inference on new tasks. Optimization-based methods [1, 28, 81, 107] learn a single model initialization that is amenable to fast adaption and can be applied to any arbitrary architecture. However, most existing metric-based and optimization-based methods assume that a single metric model or parameter initialization is sufficient to capture the entire task distribution.

5.3.2 Heterogeneous meta-learning

Heterogeneous meta-learning (HML) is an emerging area that develops meta-learning techniques that can generalize well to tasks drawn from a multi-modal distribution. Existing HML approaches account for task heterogeneity via (a) explicitly maintaining several local meta initializations (i.e., task clusters), to which observed tasks are assigned during training [105, 112], and/or (b) modulating a global meta initialization based on the learned task embedding vector [100, 105].

Nonetheless, both tactics have several drawbacks. The effectiveness of (a) largely depends on the quality of the many heuristics employed, such as the number of clusters and the distance metric used for cluster assignment. While (b) is a more principled approach that does not require expert understanding of the task distribution, both Yao et al. [105] and Vuorio et al. [100] adopt an additive weight modulation model that is extremely expensive to learn with parameter-heavy predictor architectures. To improve the efficiency of (b), this paper instead adopts a permutative routing modulation model, which implicitly maps task modes to various shuffling configurations of convolution channels in the network architecture.

5.3.3 Routing neural networks

Routing neural networks or neural routing refers to a technique in neural network architecture where information is selectively passed between groups of neurons based on some learned decision rules. This can be accomplished through the use of routing algorithms or specialized routing

layers in a neural network. The most common form of routing is by pruning computational paths (e.g., setting certain weights to zero), which is typically used to induce sparsity [91] in the network for computational efficiency, or to prevent catastrophic forgetting in continual learning scenarios [16].

Random channel shuffling was introduced by [108] in the context of designing compact architecture to improve model expressiveness. The ShuffleNet architecture was subsequently extended to explicitly learn the shuffling order [67] (i.e., via optimizing for the permutation matrices that control the shuffling). Neural routing for meta learning has only been considered by the work of Cai et al. [14] in the form of heuristic pruning. We introduce a different approach that explicitly learns to modulate channel shuffling given observed task data.

5.4 Methodology

Motivated by our discussion above and previous work which established that learning a single initialization θ_* is sub-optimal when the task distribution $\mathcal{T}$ is multi-modal, we now introduce our heterogeneous meta-learning approach (Fig. 5.2). To account for task-heterogeneity, Yao et al. [105] and Vuorio et al. [100] propose to apply task-specific modulation of the base parameter θ_* as follows:

$$\mathcal{M}_*(T_i) \quad = \quad \mathrm{mo}(\theta_*, \mathcal{D}_i) - \eta \nabla_\theta \mathcal{L}_G^\dagger(\mathrm{mo}(\theta_*, \mathcal{D}_i), \mathcal{D}_i) \; , \tag{5.3}$$

where $\mathrm{mo}(\theta_*, \mathcal{D}_i)$ abstracts the modulation operator that takes the form $\mathrm{mo}(\theta_*, \mathcal{D}_i) = \theta_* \odot \psi(\mathcal{D}_i)$ in both [105] and [100], $\odot$ denotes the point-wise multiplication operator, and ψ denotes some arbitrary embedding protocol that maps a task dataset to the weight space of the predictor G. For example, Vuorio et al. [100] models ψ as an attention mechanism, whereas Yao et al. [105] pretrains a ProtoNet task embedding [92] for task clustering and applies a fully connected network to generate $\psi(D_i)$ from the cluster centroids. Both methods, however, suffer from high additional complexity since the output dimension of ψ is the prohibitively large number of parameters in

G. To work around this shortcoming, we instead apply the task-specific modulation tactic on the

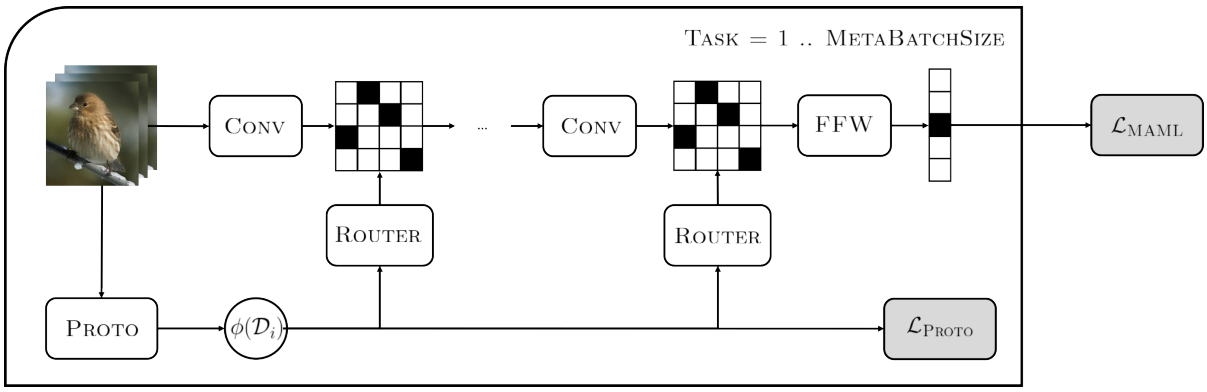


Figure 5.2: The overview of our meta-routing approach (MRM). Task data is first embedded as $\phi(\mathcal{D}_i)$. $\phi(\mathcal{D}_i)$ is then used to compute prototypical network loss [92] and generate channel routing matrices interleaving convolutional layers of the predictor network. Our loss function combines the MAML loss (over a meta-task batch) and the prototypical network loss (per task).

architecture routing level:

$$\mathcal{M}_*(T_i) \;\; = \;\; \theta_* - \eta \nabla_\theta \mathcal{L}^\dagger_{\text{mo}(G,\mathcal{D}_i)}(\theta_*, \mathcal{D}_i) \; . \tag{5.4}$$

To concretely describe our modulation operator, we will first re-imagine the architecture G as a sequence of neural layers $\{G_1, G_2, \ldots, G_M\}$, such that for any arbitrary input $\mathbf{x}$, we can rewrite $G(\mathbf{x}) = G_M \circ G_{M-1} \circ \cdots \circ G_1(\mathbf{x})$. We assume that the output of layer G_i has dimension $C_i \times d_i$, where C_i is the number of feature channels and d_i is the (flattened) feature dimension of each channel. Then, our modulation operator can be succinctly applied through interleaving a sequence of routing layers $\{R_1, R_2, \ldots, R_{M-1}\}$ in between the predictor layers of G, that is:

$$\text{mo}(G, \mathcal{D}_i) \;\; = \;\; G_M \circ R_{M-1} \circ G_{M-1} \circ \cdots \circ R_1 \circ G_1 \; , \tag{5.5}$$

where each routing layer takes the form of $R_j(Q; \mathcal{D}_i) \triangleq P_j(\mathcal{D}_i)Q$ for some intermediate feature $Q \in \mathbb{R}^{C_j \times d_j}$, such that P_j maps $\mathcal{D}_i$ to a $C_j \times C_j$ permutation matrix. Intuitively, the goal of P_j is to re-route the information flow between layer G_j and G_{j+1} of the predictor net in response

to the current task. It is thus appropriate for P_j to generate a permutation matrix, such that the information channels are shuffled without degrading their signals.

To construct such a map, we first compute the ProtoNet embedding [92] of $\mathcal{D}_i$ and apply a convolutional layer which subsequently transforms this embedding into a general square matrix in $\mathbb{R}^{C_j \times C_j}$. To approximate the discrete permutation constraint, we could directly apply a Gumbel-Sinkhorn layer [73], whose output is guaranteed to converge to a permutation matrix in the limit of its temperature parameter. We direct readers to the appropriate references for the specific details of the involved layers. However, we note that the Gumbel-Sinkhorn layer does not scale well with the total number of channels in G, and will show in the next section that the permutation constraint can be approximated more compactly using a novel layer that we call Gumbel-Beneš. For convenience, we further let π denote the combined weights of $\{P_1, P_2, \ldots, P_{M-1}\}$, which fully specifies our modulation network. The base predictor parameters θ_* and the modulation parameters γ_* can then be jointly optimized via extending the MAML loss function [28] as follows:

$$(\theta_*, \pi_*) = \underset{\theta,\pi}{\operatorname{argmin}} \, \mathbb{E}_{T_i} \left[\mathcal{L}^{\dagger}_{\mathrm{mo}(G,\mathcal{D}_i;\pi)} \left(\theta - \eta \nabla_\theta \mathcal{L}^{\dagger}_{\mathrm{mo}(G,\mathcal{D}_i;\pi)} \left(\theta, \mathcal{D}_i^t \right), \mathcal{D}_i^v \right) + \lambda \mathcal{L}_{\mathrm{Proto}}(\mathcal{D}_i; \pi) \right] \quad (5.6)$$

where $\mathcal{L}_{\mathrm{Proto}}$ denotes the PROTONET loss [92] and λ denotes the trade-off hyper-parameter.

5.4.1 Gumbel-Beneš Routing Layer

The Gumbel-Sinkhorn layer is a differentiable transformation that approximately produces permutation matrices. However, in order to generate a sparse permutation matrix of size $C \times C$, it is necessary that the Gumbel-Sinkhorn layer also receives as input a dense $C \times C$ matrix. Due to this requirement, each routing layer R_j will require at least a quadratic number of parameters in terms of C_j (e.g., for the convolutional map) to generate an input to the Gumbel-Sinkhorn layer. Overall, the addition of the entire modulation component would yield an extra $\mathcal{O}(\sum_{j=1}^{M} C_j^2)$ learnable parameters. Although this additional complexity would be smaller than the total number of parameters in G in most cases, it could become very expensive for larger architectures.

To overcome this challenge, we first point to a classical result from Beneš [9], which shows that any permutation of C channels can be emulated by exactly $2\log_2 C$ layers of $C/2$ binary switches (e.g., 2×2 permutation matrix). Within each layer, we divide the input channels into groups of two and permute their feature pairwise via multiplying with the content of the binary switches. The output of one layer is then forwarded to the next via a deterministic *exchange step* [9]. An example routing configuration is shown in Fig. 5.1. It was shown that the Beneš network has a congestion of 1 [9], meaning there is no two different permutations that share the same switch configurations. As a result, it is sufficient to compactly model channel routing with just $C_j \log_2 C_j$ parameters at any layer R_j, instead of C_j^2 if the Gumbel-Sinkhorn layer were employed.

Finally, we can redefine our routing layer R_j in terms of the Beneš network. This is achieved by first reformulating the convolutional map to produce a stacked tensor of continuous switch configurations. We then apply the Gumbel-Sinkhorn transformation on these continuous configurations to approximate the discrete binary switches. Finally, we perform the shuffle-exchange steps recursively to permute the input channels. Formally, we describe the computational pathway of the routing layer R_j given input Q and task data $\mathcal{D}_i$ as follows:

$$
\begin{aligned}
S_0 &= Q \\
U_j &= f_j(\boldsymbol{\phi}(\mathcal{D}_i)) \\
\begin{bmatrix} \hat{S}_l[2k-1] \\ \hat{S}_l[2k] \end{bmatrix} &= \mathrm{GS}(U_j[l,k]) \begin{bmatrix} S_l[2k-1] \\ S_l[2k] \end{bmatrix} \quad \forall k \in [1, C_j/2] \\
S_{l+1} &= \mathrm{exchange}\left(\hat{S}_l\right) \\
R_j(Q, \mathcal{D}_i) &= S_{2\log_2 C}\ ,
\end{aligned}
\tag{5.7}
$$

where $\boldsymbol{\phi}$ denotes the PROTONET embedding, f_j maps task embedding to continuous switch configurations, $\mathrm{GS}(U_j[l,k])$ denotes the Gumbel-Sinkhorn transform of the component of U_j corresponding to the k^{th} switch of l^{th} layer. The exchange step refers to the deterministic routing of Beneš network, which we refer readers to Beneš [9] for details. Last, each S_l denotes the

output of the l^{th} Beneš layer.

5.5 Empirical Study

We compare the performance of our method using the Gumbel-Sinkhorn permutation layer [73] (MRM-GS) and our proposed Gumbel-Beneš routing layer (MRM-GB) against several meta-learning baselines, including MAML [28], its first-order approximation FO-MAML [1], the prototypical network method (PROTONET) [92] and the multi-modal model-agnostic meta learning method (MMAML) [100]. We adapt the MAML and PROTONET implementations from the LEARN2LEARN package [2], and implement the rest in PyTorch. Experiments are conducted on a GTX-3080 GPU with 13GB memory.

For MRM-GS, MRM-GB and MMAML, we parameterize the task embedding network and the predictor network with the same CNN architecture with 4 hidden convolutional blocks and a simple feed-forward classification layer. Each block consists of a 3×3 convolution layer, followed by BATCHNORM, MAXPOOL and RELU activations. All convolution layers have $C = 32$ or 64 hidden channels, depending on the specific task distribution. The mapping from task embedding to modulation parameters is parameterized by a 1-layer, TANH-activated feed-forward neural network, whose output dimension depends on the method (e.g., approximately C^2 for MRM-GS, $C \log_2 C$ for MRM-GB and $9C^2$ for MMAML). We apply the modulation to the first hidden convolutional layer.

Both MAML and FO-MAML have no embedding network. For fair comparison against the above methods, we parameterize the predictor network by a two-headed CNN architecture with 4 hidden convolutional blocks per head. Their outputs are then concatenated and forwarded to the classification layer for prediction. Last, PROTONET has no predictor network and performs prediction via clustering the input embeddings. For the same reason as above, we parameterize its embedding network by a similar two-headed CNN architectures (no classification layer). Our experiments are conducted on several meta-learning vision baselines, described as follows:

- The OMNIGLOT dataset [56], which consists of 1623 handwritten characters from 50 different alphabets and writing systems. We randomly split the dataset by class into train (1100 classes), validation (100 classes), and test sets (423 classes), as suggested by [84].
- The MINI-IMAGENET dataset [99], which is a subset of the larger ImageNet dataset [87] that contains 60000 images from 100 object categories. We randomly split the dataset by category into train (64 categories), validation (16 categories), and test sets (20 categories).
- The JIGSAW-OMNIGLOT and JIGSAW-MINI-IMAGENET datasets, which are obtained by segmenting the training images in the respective original datasets into 2×2 tiles and randomly permuting these tiles to simulate 24 different task modalities.
- The FLOWER-AIRCRAFT-FUNGI dataset, which combines: (a) The VGGFLOWER102 dataset [97] consisting of 102 classes of flowers (between 40 to 258 images per class); (b) the FGVCAIRCRAFT dataset [68, 97] consisting of 102 classes of aircraft (100 images per class); and (c) the FGVCFUNGI dataset [97] consisting of 1394 classes of fungi, with a total of 89760 images.

5.5.1 Meta-learning for uni-modal task distribution

We first show that our method performs robustly on the traditional homogeneous meta-learning setting despite the multi-modal treatment. We train all baseline methods on random batches of tasks drawn from (a) the OMNIGLOT dataset; and (b) the MINI-IMAGENET dataset. All tasks consist of randomly drawn images from 5 distinct labels. For each label, the task dataset contains n_s support and n_q query images. For training, both the support and query images are used to train the meta-learners. For testing, we perform fast adaptation using the support image and measure the test accuracy on the query images. For each epoch, we sample a batch of 32 training tasks to train each baseline method and evaluate their averaged performances over 5 random test tasks. For the OMNIGLOT experiments, $n_s = 1$, $n_q = 15$. For the MINI-IMAGENET experiments, $n_s = 5$, $n_q = 5$.

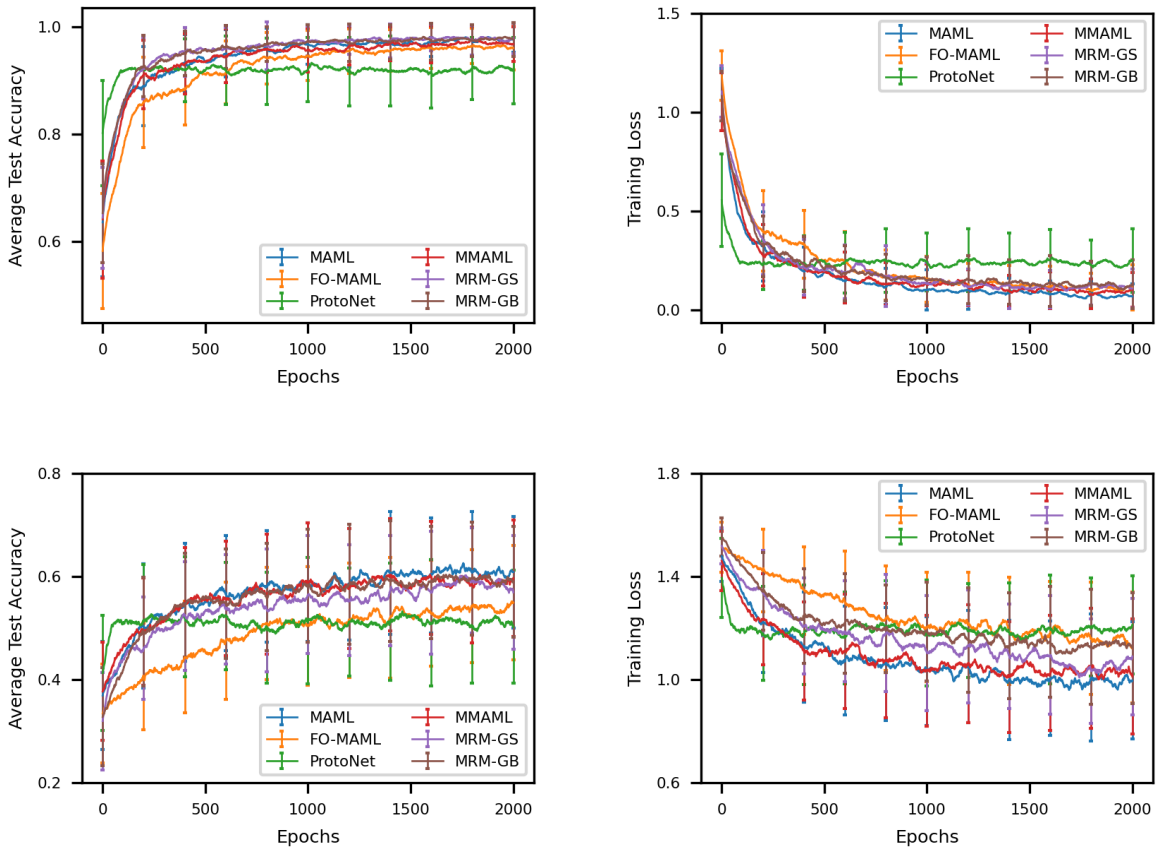


Figure 5.3: Plots of training loss and average test accuracy vs. number of training epochs for various baseline methods on OMNIGLOT (top row) and MINI-IMAGENET dataset (bottom row). Error bars are obtained over 5 random test task batches.

Fig 5.3 plots the running mean (over 50 consecutive epochs) of (a) average test accuracy (over 5 test tasks); and (b) training loss against the number of training epochs for each baseline method. Across both datasets, we observe a gradual decrease in training loss and increase in test accuracy over 2000 training epochs for all methods, including our meta-routing frameworks MRM-GB and MRM-GS.

Additionally, MAML, MMAML, MRM-GS and MRM-GB all converge to a similar test accuracy on this unimodal task (e.g., approximately 0.98 for OMNIGLOT and 0.60 for MINI-IMAGENET). The final classification accuracies of both MRM-GS (0.978/0.597) and MRM-

GB ($0.976/0.590$) are comparable to MAML ($0.977/0.609$) on these unimodal settings, while significantly outperforming FO-MAML ($0.960/0.552$) and PROTONET ($0.918/0.502$). This is expected since FO-MAML uses an inaccurate first-order approximation of the MAML loss, whereas PROTONET does not make use of the task diversity in the meta task batch. We will omit the results of FO-MAML in subsequent experiments because it is consistently outperformed by MAML in every scenario. Overall, our empirical results demonstrate that the proposed meta-routing framework performs robustly in standard meta-learning setting despite the extra consideration for task heterogeneity.

5.5.2 Meta-learning for multi-modal task distribution

2×2 Jigsaw Omniglot and Jigsaw Mini-Imagenet

We further conduct experiments to demonstrate the performance of our method in two different settings of task heterogeneity. In the first experiment, we simulate the multi-modality of the task distribution by applying a *jigsaw* transformation to the training images in the OMNIGLOT and MINI-IMAGENET datasets. Specifically, each training/test image is first segmented into 2×2 smaller tiles. For each sampled task, we then randomly draw a permutation of these 4 tiles and shuffle them accordingly to systematically derive new tasks that belong to $4! = 24$ different modalities.

Fig. 5.4 plots the running mean (over 50 consecutive epochs) of (a) average test accuracy (over 5 test tasks); and (b) training loss against the number of training epochs for each baseline method. We perform a total of 3000 training epochs for each JIGSAW-OMNIGLOT experiment, and 5000 training epochs for each JIGSAW-MINI-IMAGENET experiment, which are sufficient to observe convergence.

PROTONET demonstrates poor learning on these task distributions. This is most likely because PROTONET tries to assign similar embeddings to images of the same label, which include different jigsaw shufflings of the same image. While our approaches also make use of

the PROTONET loss to embed tasks, the specificity of the shuffling will be captured by meta-training the predictor network and the modulator network using the MAML loss. As a result, our methods MRM-GB $(0.944/0.595)$ and MRM-GS $(0.948/0.580)$ consistently achieve the best average test accuracies in both datasets. On the harder JIGSAW-MINI-IMAGENET dataset, our baselines improve between $5.8 - 8.5\%$ over the classification performance of MAML. The other multi-modal baseline MMAML $(0.935/0.580)$ outperforms MAML on the JIGSAW-MINI-IMAGENET, but performs poorly on JIGSAW-OMNIGLOT.

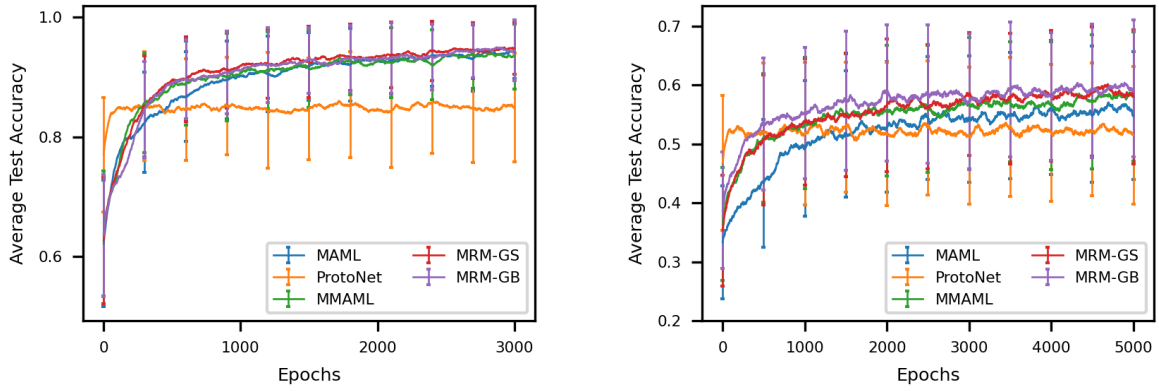


Figure 5.4: Running means of average test accuracy vs. number of training epochs for various baseline methods on the JIGSAW-OMNIGLOT (left) and JIGSAW-MINI-IMAGENET dataset (right).

Fig. 5.5 further plots the running means of training loss (over 50 consecutive epochs) vs. number of training epochs for various baseline methods on the 2×2 JIGSAW-OMNIGLOT and JIGSAW-MINI-IMAGENET datasets. As expected, we observe trends similar to the classification accuracy plots shown in Fig. 5.4. Specifically, the training loss of our methods, MRM-GS and MRM-GB, converge to comparable values on the easier JIGSAW-OMNIGLOT dataset, and outperform other baselines by 7% on the more complex JIGSAW-MINI-IMAGENET dataset.

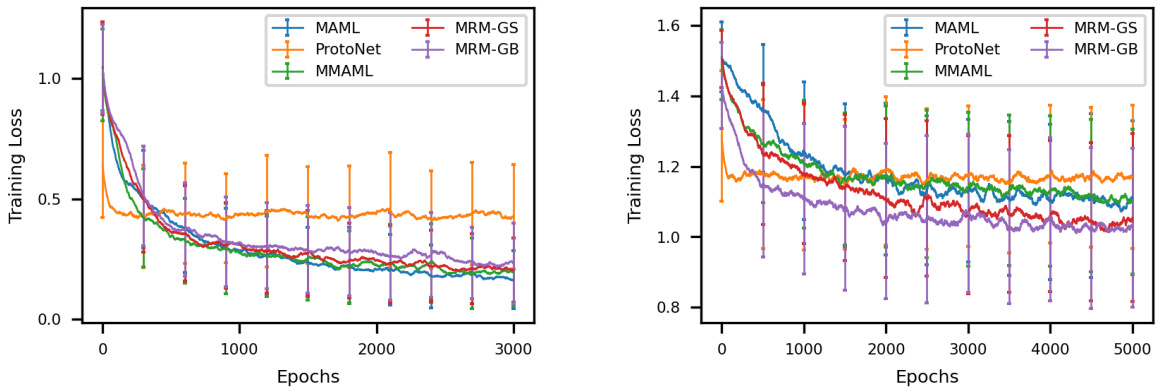


Figure 5.5: Running means of average training loss vs. number of training epochs for various baseline methods on the JIGSAW-OMNIGLOT (left) and JIGSAW-MINI-IMAGENET dataset (right). The vertical bars denote the standard deviations over 5 random test tasks.

5.5.3 4x4 Jigsaw Mini-Imagenet

In this experiment, we increase the number of modalities of the JIGSAW MINI-IMAGENET task distribution. This is achieved by adopting a finer segmentation of training images into 4×4 jigsaw pieces. Fig. 5.6 below plots the running means of training loss and classification accuracy of various baselines. Here, we drop both PROTONET and FO-MAML, as we have demonstrated that they are sub-optimal baselines for multi-modal scenarios. We observe that both MRM-GS and MRM-GB are clearly more robust in this setting, having significantly outperformed MAML and MMAML in both accuracy (18% increase) and training loss (17% decrease). This result suggests that our meta routing scheme is most suitable for complex task distributions.

Flower-Aircraft-Fungi

In this experiment, we simulate multi-modality via grafting three different image datasets: (a) VGGFLOWERS102, (b) FGVCAIRCRAFT and (c) FGVCFUNGI. The combined FLOWERS-AIRCRAFT-FUNGI dataset thus has three distinct task modalities. We initialize the task distribution with only data from (a) and subsequently inject data from (b) after 16000 sampled train/test tasks (around

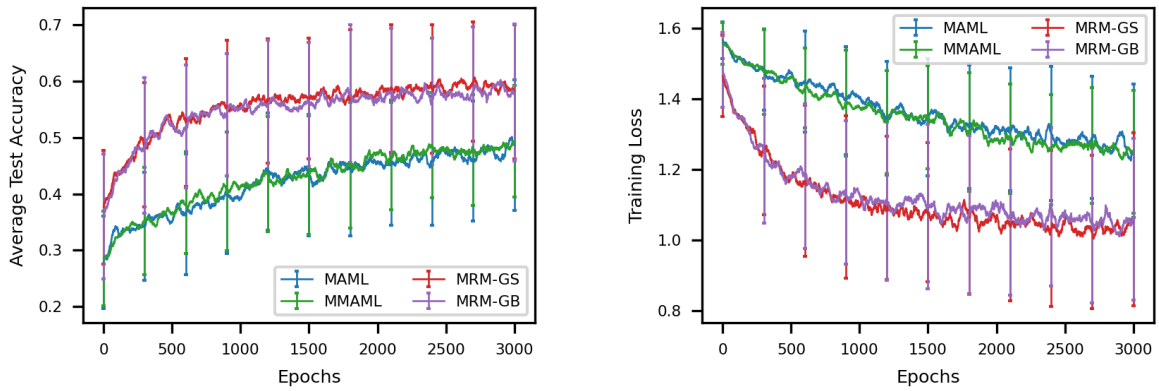


Figure 5.6: Running means of average accuracy and training loss vs. number of training epochs for various baseline methods on the 4×4 JIGSAW-MINI-IMAGENET dataset. The vertical bars denote the standard deviations over 5 random test tasks.

1230 epochs); and from (c) after 32000 sampled train/test tasks (around 2460 epochs). For this experiment, we use a batch size of 8 tasks per epoch instead of 32 like the above experiments.

Fig. 5.7 plots the running mean of average test accuracies (over 5 test tasks) for the same meta-learning baselines. The running mean window is increased to 100 to ensure visual clarity. The dotted vertical lines in each plot mark the injection points of subsequent datasets into the task streams. As expected, each task injection causes a significant drop in average test accuracy (and likewise, a spike in training loss) due to the introduction of a new modality in the distribution. This confirms that the meta initialization learned on one modality cannot be easily adapted to address tasks from another modality, and thereby motivates the need to address task-heterogeneity in meta-learning. Last but not least, MRM-GB achieves the best average test accuracy (0.342) among all baselines, improving 2.4% over MMAML (0.334) and 8.9% over MAML (0.314). This result corroborates our previous observations regarding the robust performance of our approach in a multi-modal setting.

Finally, we simulate the scenario where all three datasets are simultaneously injected onto the task stream at the very first training epoch. In this way, the meta-model is not given sufficient training budget to learn each dataset individually, and has to contrast the heterogeneous learn-

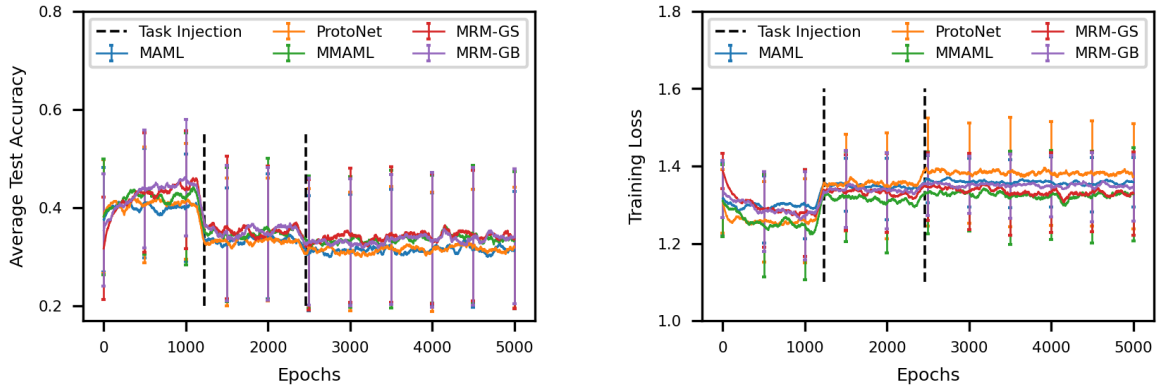


Figure 5.7: Plots of running means of training loss and average test accuracy vs. number of training epochs for various baseline methods on the multi-modal FLOWERS-AIRCRAFT-FUNGI dataset. The dotted vertical lines respectively mark the injection points of the FGVCAIRCRAFT and the FGVCFUNGI datasets into the task distribution.

ing signals at every point of the training process. Fig. 5.8 plots the running means of training loss and test accuracy for the same baselines above in this scenario. As expected, the standard deviations of training loss and test accuracy are generally larger than in the original setting with sequential injection points. More importantly, we again observe that both MRM-GB and MRM-GS outperform MAML and MMAML, thus confirm the effectiveness of our method on highly heterogeneous task streams.

5.5.4 Scalability of the modulation networks

Last, we show that our method scales well in terms of the predictor network's architecture complexity. Specifically, we vary the number of hidden channels C per convolutional block in our standard 4-block CNN architecture described above and record the training time per epoch as well as the number of modulating parameters in Table 5.1. For this experiment, weight modulation (MMAML) and routing modulation (MRM-GB, MRM-GS) are applied to all 4 blocks. The reported runtimes are averaged over 200 consecutive training epochs.

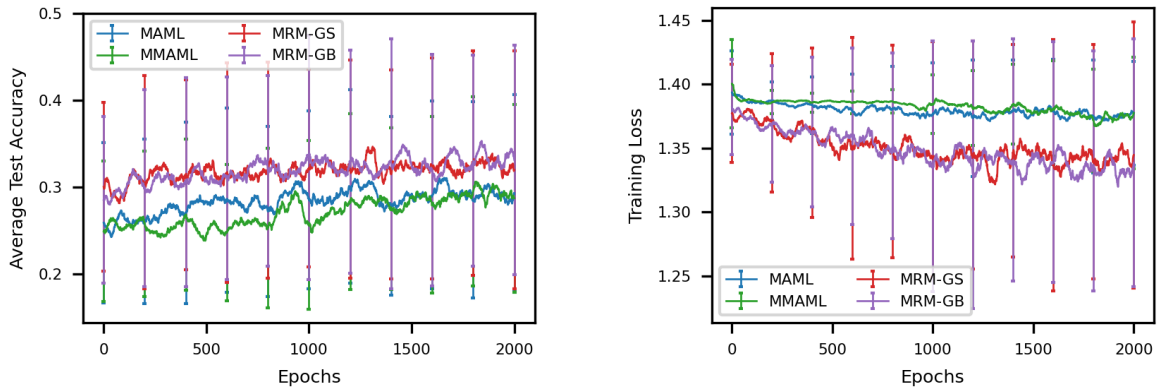


Figure 5.8: Plots of running means of training loss and average test accuracy vs. number of training epochs for various baseline methods on the multi-modal FLOWERS-AIRCRAFT-FUNGI dataset. All three datasets are injected at epoch 0. The vertical bars denote the standard deviations over 5 random test tasks.

As expected, the number of parameters in MRM-GB increases almost linearly in C, resulting in the most efficient runtime at every size of the predictor architecture. We note that the increase in runtime, however, does not always correlate perfectly to the increase in the number of modulation parameters. This is due to various other factors, such as the overhead cost of computing the more complicated gradients of MRM-GB, MRM-GS compared to MMAML. Nonetheless, as C becomes very large, we observe that the number of modulation parameters in both MRM-GS and MMAML will grow prohibitively expensive. For example, at $C = 256$ (which is reasonable for many common architectures), we were no longer able to fit the MMAML model into the GPU memory.

5.6 Conclusion

Existing meta-learning methods assume a homogeneous task distribution, which limits their generalization ability when dealing with multi-modal task distributions. Several recent works have attempted to rectify this problem, but suffer from increased complexity in terms of parameters.

Table 5.1: Runtime (in seconds) per training epoch and number of learnable parameters in the modulation networks of MRM-GB, MRM-GS and MMAML at different numbers of hidden convolution channels. — indicates that the model exceeds our GPU memory, and its runtime cannot be recorded.

METHOD	MRM-GB		MRM-GS		MMAML	
C	RUNTIME	PARAMS	RUNTIME	PARAMS	RUNTIME	PARAMS
16	3.25 ± 0.33	0.04M	3.35 ± 0.36	0.08M	3.83 ± 0.41	0.75M
32	3.57 ± 0.07	0.21M	4.48 ± 0.58	0.66M	6.36 ± 0.37	5.93M
64	4.21 ± 0.08	0.99M	5.58 ± 0.38	5.26M	7.96 ± 0.38	47.3M
128	5.06 ± 0.08	4.59M	7.16 ± 0.59	42.0M	12.91 ± 0.79	378M
256	7.84 ± 0.34	21.0M	12.64 ± 0.57	336M	—	3.02B

To overcome this limitation, we propose an innovative strategy for heterogeneous meta-learning. Our approach efficiently captures the multi-modality of the task distribution by modulating the routing between convolution channels in the network. This mechanism can be viewed as a permutation learning problem, which we model and solve using a compact neural permutation layer based on the classical Benes routing network.

Our Gumbel-Benes layer exhibits sub-quadratic parameter complexity in the total number of channels, in contrast to the quadratic complexity of state-of-the-art Gumbel-Sinkhorn layers. We validate our approach on various multi-modal meta-learning benchmarks, demonstrating superior performance compared to previous methods in terms of both generalization accuracy and runtime. Theoretical understanding of our approach will be a future research consideration.

Chapter 6

Conclusion and Future Directions

New algorithmic advances in Machine Learning (ML) and Computational Biology (CB) have surpassed human capabilities in tasks previously considered too complex for computers. However, to construct computational pipelines with impressive predictive power, the practical design of the solution needs to be carefully calibrated for every task instance. This process is complex, iterative, and frequently involves human trial-and-error effort, making it time-consuming and computationally expensive. The study of automated algorithm design (AAD) aims to revolutionize the way these applications are developed by automating tedious tasks such as hyper-parameter optimization and architecture search, leading to improved accuracy and efficiency. This dissertation explores various aspects of the AAD problem in both ML and CB applications.

Chapter 2 investigates the kernel selection (KS) problem for kernel-based regression methods that are widely applied in many analytic applications. This is a challenging problem because the kernel function search space is complex and arbitrarily large, yet any valid kernel function must satisfy a semi positive definite constraint. Previous selection methods heuristically restrict the search space to finite sets of candidate functions to enable tractable search algorithms, but this strategy may limit the diversity of potential kernels. To overcome this problem, we propose the first kernel search algorithm for the infinitely large composite kernel search space, induced by a generative grammar and a vocabulary of atomic functions. We approach the unboundedness of

this search space by casting the kernel search problem as simultaneously optimizing a recurrent generative model and a self-supervised termination policy that prevents the generation of overly complex and impractical kernels. To achieve this, we develop a bi-level Bayesian optimization technique to alternately optimize these two components of the reparameterized objective. Our algorithm outperforms many existing baseline methods on different kernel recovery and regression kernel search tasks.

However, our kernel selection approach is limited to composite kernels built from a predefined set of atomic functions. Implicitly, it assumes that the most suitable kernel expression can be represented as a combination of the known kernels. Such assumption lacks expressiveness and may restrict the method's usability in some applications. To overcome this limitation, a potential extension is to incorporate recent advances in deep kernel learning [102], which utilizes deep neural networks to parameterize the kernel functions instead of predefined atomic functions. In particularly, a deep kernel k_ϕ can be written as:

$$k_\phi(\mathbf{x}_i, \mathbf{x}_j; w_\phi, w_b) \quad \triangleq \quad k_b\left(\phi(\mathbf{x}_i; w_\phi), \phi(\mathbf{x}_j; w_\phi); w_b\right) \ , \tag{6.1}$$

where k_b denotes some known kernel function, ϕ denotes a neural feature encoder, and w_b, w_ϕ are respectively their learnable parameters. This converts the problem of searching for optimal composite kernel expressions into an architecture search task for the encoder network ϕ, which can leverage existing methods described in Chapter 4. As the neural network search space is significantly more expressive compared to heuristic choices of the base kernel vocabulary, this reformulation may lead to the discovery of more expressive neural kernels, hence offering enhanced flexibility and performance for a wider range of tasks.

Chapter 3 investigates the minimizer sketch design (MSD) problem for biological sequence compression. In particular, we study the family of minimizer sketching schemes that are parameterized by k-mer permutations. This combinatorial parameter space makes minimizer sketches highly difficult to optimize with respect to the various sketching goals. As such, all previous works have turned to heuristic surrogate objectives with limited theoretical connection to the

original metrics, and thus they only achieve moderate performance on realistic sketching tasks. Our approach for both minimizer variants presented in this dissertation is to train a deep convolutional neural scoring function that implicitly represents a k-mer permutation. We arrive at a tractable learning objective via further introducing a self-supervised template model that captures the characteristics of k-mer permutations that are likely to induce desirable sketches. Via aligning the output of the score function to that of the template function, we arrive at the first differentiable objectives for minimizer optimization, yielding significantly improved run-time and sketching performance on many biological sequence baselines.

The MSD frameworks investigated in this thesis are limited to the representation of sequence sketches as collections of fixed-length k-mers. A potential extension to this dissertation is to learn sketches that comprise of variable-sized substrings, or continuous latent embeddings of substrings. The variable-sized substrings perspective has recently been considered by Sahlin [88], whose method achieves promising performance on many alignment baselines. However, there is much more to achieve in this direction, as Sahlin [88] simply relies on a heuristic procedure to link together clusters of densely sampled syncmers [24]. The latent embedding perspective, on the other hand, aligns with the idea of learning useful representations for natural language tokens [21], which recently extends to biological domains [13, 50]. Nonetheless, these pre-trained embedding models aim to learn representations that are useful for prediction, and thus do not support sequence alignment. As such, it will be challenging to configure these embeddings in ways that allow us to infer conserved domains, substitutions and deletions between sequences.

Another venue for future work is extending the proposed minimizer optimization algorithms to the local scheme [71], which implements different local k-mer orderings for different windows (in contrast to applying a single global ordering to all windows, such as in the minimizer approach). This modification would potentially yield sketching schemes that are more robust in long sequences with multiple distinct local patterns. Optimizing the local scheme translates to

learning a neural scoring function f that is consistent on a window level, but not on a k-mer level. That is, similar windows will rank constituent k-mers similarly, but a k-mer may not receive the same score everywhere in the sequence. Because of this, there is no single score vector that represents the entire sequence, and thus it would be challenging to design template functions that capture desirable scoring patterns. Modelling a new class of template functions would therefore be the central goal of this research direction.

Chapter 4 investigates the federated neural architecture search (FNAS) problem. The goal is to find optimal neural network architectures for a collection of decentralized clients seeking to solve related computational tasks. This setup is especially useful in privacy-sensitive (e.g., clients do not want to share data) and resource-constrained (e.g., clients do not have sufficient data) systems as it allows high-performing architectures to be collectively inferred. Previous works in this direction are restricted to the scenario in which all clients seek to solve the same task, and therefore they focus on finding a homogeneous architecture across clients. This dissertation instead considers a more realistic personalized learning setting where local tasks are not necessarily identical, and might require architectural adaptations to address task-specific nuances. We approach this challenge from a meta-learning perspective, with the goal of designing a base architecture that can efficiently adapt to specific tasks with reasonably small fine-tuning budgets. To facilitate this, we develop a novel search space for personalizable architectures, and a new approximation of the MAML meta learning objective [28] to ensure the scalability of our algorithm. Finally, we demonstrate that our method, FEDPNAS, outperforms the federated extension of previous state-of-the-art NAS frameworks on various vision benchmarks.

As the proposed approach brings together Federated Learning, Meta Learning and NAS, it naturally inherits the technical challenges from each approach. For example, the learning degradation issue for heterogeneous clients in FL [60] is substantially more challenging when applied to NAS due to the large dimension of the over-parameterized master network. Our heterogeneous learning framework in Chapter 5 is a step toward addressing the task heterogeneity challenge,

but it does not trivially extend to the NAS setting and thus this challenge will remain an avenue for future investigations.

In addition, existing FL methods, including the proposed FEDPNAS approach, typically assume that the solution model is fully parametric and can be jointly inferred via aggregating the local copies of its parameters. This mode of aggregation is not applicable to non-parametric models, such as the Gaussian process regression [83] framework, that compute data-dependent predictions of the form $y_* = f(\mathbf{x}_*, \mathcal{D})$, where $\mathbf{x}_*$ denotes the unseen input, $\mathcal{D}$ denotes the observed training data, and y_* denotes the predicted outcome corresponding to $\mathbf{x}_*$. This is because the aggregated global model would necessarily require access to the local data to perform prediction, and thus it will trivially violate the privacy-preserving objective of FL. A potential approach to this problem is to segment the solution architecture into distinct communicable and non-communicable portions (in an analogous fashion to the FEDPNAS search space) to facilitate partial model aggregation. For example, a Gaussian process with deep kernel [102] can be made amenable to FL via communicating the parameters of the encoders while keeping the local data private. It would also be of interest to investigate the convergence of this approach as compared to that of traditional FL with full model aggregation.

Chapter 5 investigates the meta-learning problem that focuses on training models to quickly adapt and generalize to unseen tasks with minimal data. Meta-learning has found applications in various fields, including computer vision, natural language processing, robotics, and optimization problems. It holds the promise of enabling artificial general intelligence and potentially reduces the need for large amounts of task-specific data and enhancing the efficiency of learning algorithms.

Most existing meta-learning frameworks operate under the assumption that the optimal solution distribution (associated with tasks within the task distribution) is concentrated around a singular mode, and thus they deem it sufficient to train a single base model that can easily adapt to any other task. Unfortunately, this assumption does not encompass situations where the task

distribution is multi-modal. To tackle this restrictive assumption, this dissertation introduces a new class of convolutional neural architecture that incorporates adaptive channel routing layers to implicitly represents exponentially many solution modes (architecture configurations). By learning a task-driven routing model, our method maps similar tasks to the same mode and different tasks to different modes. Consequently, for unseen tasks, fine-tuning is only necessary to adjust the nearest mode toward the optimal solution. We show that this approach results in a more efficient meta-learning protocol that outperforms previous state-of-the-art methods on several multi-modal baseline task streams.

While this dissertation provides a new algorithmic perspective on solving the heterogeneous meta-learning problem, there remain several promising avenues for future exploration. One crucial direction is to investigate a more rigorous quantification of task heterogeneity. Developing robust measures to gauge and categorize the diversity across tasks could provide a foundation for designing tailored solutions that effectively accommodate varying data distributions, modalities, and complexities. Furthermore, a weakness of the current framework is that even though many different solution modes are implicitly encoded in the channel routing architecture, they are strongly correlated due to having shared parameters, and thus are not ideal for generalization to unseen task modalities. A crucial future direction therefore involves the development of more robust meta-models that yield decoupled solution modes, while maintaining efficiency during the training process.

Bibliography

[1] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. *arXiv preprint arXiv:1810.09502*, 2018. 5.1, 5.3.1, 5.5

[2] Sébastien M R Arnold, Praateek Mahajan, Debajyoti Datta, Ian Bunner, and Konstantinos Saitas Zarkias. learn2learn: A library for Meta-Learning research, August 2020. URL `http://arxiv.org/abs/2008.12284`. 5.5

[3] Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In *Conference on Learning Theory*, pages 213–274. PMLR, 2017. 1.2

[4] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018. 1.2

[5] Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 919–932, 2021. 1

[6] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., 1998. 1.1.2

[7] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli.

Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, 65:83–92, 2016. 1.1.1

[8] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 550–559. PMLR, 2018. 1.2, 4.3, 4.3.2

[9] Václav E Beneš. Permutation groups, complexes, and rearrangeable connecting networks. *Bell System Technical Journal*, 43(4):1619–1640, 1964. 5.1, 5.4.1, 5.4.1

[10] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2), 2012. 1.1.1

[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011. 1.1.1

[12] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1): 10–15, 1993. 1.1.1

[13] Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, and Michal Linial. ProteinBERT: a universal deep-learning model of protein sequence and function. *Bioinformatics*, 38(8):2102–2110, 2022. 6

[14] Jicang Cai, Saeed Vahidian, Weijia Wang, Mohsen Joneidi, and Bill Lin. Neural routing in meta learning. *arXiv preprint arXiv:2210.07932*, 2022. 5.3.3

[15] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016. 3.1, 3.2.3, 3.3

[16] Mark Collier, Efi Kokiopoulou, Andrea Gesmundo, and Jesse Berent. Routing networks with co-training for continual learning. *arXiv preprint*, 2020. 5.3.3

[17] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20

(3):273–297, 1995. 1

[18] Dan DeBlasio, Fiyinfoluwa Gbosibo, Carl Kingsford, and Guillaume Marçais. Practical universal k-mer sets for minimizer schemes. In *Proceedings of the 10^{th} ACM Conference on Bioinformatics, Computational Biology*, pages 167–176. Association for Computing Machinery, 2019. 3.1, 3.2.3, 3.3

[19] Dan Deblasio, Kwanho Kim, and Carl Kingsford. More accurate transcript assembly via parameter advising. *Journal of Computational Biology*, 27(8):1181–1189, 2020. 1.1.1

[20] Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, et al. KMC 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015. 3.1

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv Preprint*, 2018. 6

[22] Abhinav Dutta, David Pellow, and Ron Shamir. Parameterized syncmer schemes improve long-read mapping. *PLoS Computational Biology*, 18(10):e1010638, 2022. 3.1, 3.2.1, 3.2.2, 3.4, 3.4.2, 3.4.5

[23] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and G. Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1166–1174, 2013. 1, 1.2, 1.3, 2.1, 2.2, 2.3, 2.4

[24] Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, 2021. 1.3, 3.1, 3.2.1, 6, 3.2.2, 8, 3.4, 3.4.5, 3.4.5, 6

[25] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics, 407-499*, 2004. 2.4, 2.4.2

[26] Barış Ekim, Bonnie Berger, and Yaron Orenstein. A randomized parallel algorithm for efficiently finding near-optimal universal hitting sets. In *Proceedings of the* 24^{th} *Annual*

International Conference on Research in Computational Molecular Biology, 2020. 1.2, 3.2.3, 3.3, 3.3.4, 3.4, 3.4.2, 3.4.5, 3.4.5, 3.4.5

[27] Marius Erbert, Steffen Rechner, and Matthias Müller-Hannemann. Gerbil: a fast and memory-efficient k-mer counter with GPU-support. *Algorithms for Molecular Biology*, 12(1):1–12, 2017. 3.1

[28] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. ICML*, pages 1126–1135, 2017. 1.3, 4.4, 4.4.2, 5.1, 5.2, 5.3.1, 5.4, 5.5, 6

[29] Dan Flomin, David Pellow, and Ron Shamir. Data set-adaptive minimizer order reduces memory usage in k-mer counting. *Journal of Computational Biology*, 29(8):825–838, 2022. 3.4.5

[30] David B Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994. 1.1.2

[31] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010. 1.1.1

[32] Tatsuo Fukagawa and William C. Earnshaw. The centromere: chromatin foundation for the kinetochore machinery. *Developmental Cell*, 30(5):496–508, 2014. 3.3.4

[33] Y. Gal and R. Turner. Improving the Gaussian process sparse spectrum approximation by representing uncertainty in frequency inputs. In *Proc. ICML*, pages 655–664, 2015. 2.5

[34] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014. 5.3.1

[35] Chaoyang He, Murali Annavaram, and Salman Avestimehr. FedNAS: Federated deep learning via neural architecture search. *arXiv e-prints*, pages arXiv–2004, 2020. 1, 1.2, 4.3.4

[36] J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Proc. UAI*,

pages 282–290, 2013. 2.4, 2.4, 2.4.2, 2.5

[37] Minh Hoang and Carl Kingsford. Personalized neural architecture search for federated learning. *New Frontiers in Federated Learning: Privacy, Fairness, Robustness, Personalization and Data Ownership, Neural Information Processing Systems*, 2021. 4.1

[38] Minh Hoang and Carleton Kingsford. Optimizing dynamic structures with bayesian generative search. In *International Conference on Machine Learning*, pages 4271–4281. PMLR, 2020. 2.1

[39] Minh Hoang, Nghia Hoang, Bryan Kian Hsiang Low, and Carleton Kingsford. Collective model fusion for multiple black-box experts. In *International Conference on Machine Learning*, pages 2742–2750. PMLR, 2019. 4.6

[40] Minh Hoang, Nghia Hoang, Hai Pham, and David Woodruff. Revisiting the sample complexity of sparse spectrum approximation of gaussian processes. *Advances in Neural Information Processing Systems*, 33:12710–12720, 2020. 2.5

[41] Minh Hoang, Hongyu Zheng, and Carl Kingsford. DeepMinimizer: A differentiable framework for optimizing sequence-specific minimizer schemes. In *International Conference on Research in Computational Molecular Biology*, pages 52–69. Springer, 2022. 3.1

[42] Trong Nghia Hoang, Kian Hsiang Low, Patrick Jaillet, and Mohan Kankanhalli. Active learning is planning: Nonmyopic ε-bayes-optimal active learning of gaussian processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 494–498. Springer, 2014. 2.5

[43] Martin Hofmann. Support vector machines-kernels and the kernel trick. *Notes*, 26(3): 1–16, 2006. 2.2

[44] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. DSNAS: Direct neural architecture search without parameter retraining. In

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12084–12092, 2020. 4.1, 4.1, 4.3.3, 4.4, 4.4.1, 4.4.3, 4.5

[45] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011. 1.1.3

[46] J. Idrissi, M. Amine, R. Hassan, G. Youssef, and E. Mohamed. Genetic algorithm for neural network architecture optimization. In *International Conference on Logistics Operations Management*, pages 1–4, 05 2016. doi: 10.1109/GOL.2016.7731699. 1

[47] Chirag Jain, Arang Rhie, Haowen Zhang, et al. Weighted minimizer sampling improves long read mapping. *Bioinformatics*, 36(Supplement_1):i111–i118, 2020. 3.1, 3.2.3, 3.3

[48] Chirag Jain, Arang Rhie, Nancy Hansen, et al. Long-read mapping to repetitive reference sequences using Winnowmap2. *Nature Methods*, 19:1–6, 2022. doi: 10.1038/s41592-022-01457-8. 3.1

[49] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. *arXiv preprint arXiv:1611.01144*, 2016. 4.3.3, 4.4.1

[50] Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V Davuluri. DNABERT: pre-trained bidirectional encoder representations from transformers model for DNA-language in genome. *Bioinformatics*, 37(15):2112–2120, 2021. 6

[51] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Conference on Neural Information Processing Systems (NeuRIPS)*, 02 2018. 1.2

[52] C. D. Keeling and T. P. Whorf. Atmospheric carbon dioxide concentrations derived from flask air samples at sites in the SiO network. https://www.openml.org/d/41187, 2004. 2.4, 2.4.2

[53] D. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *Proc. ICLR*, 2013. 2.1,

2.5

[54] Diederik P. Kingma and Jimmy Ba. ADAM: A Method for Stochastic Optimization. *Computing Research Repository*, 1412.6980, 2015. 3.3.4, 3.4.5

[55] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 Dataset. Technical report, University of Toronto, 2014. URL `http://www.cs.toronto.edu/kriz/cifar.html`. 4.5

[56] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 5.5

[57] Y. LeCun, C. Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]*, 2, 2010. URL `http://yann.lecun.com/exdb/mnist`. 4.5

[58] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015. 4.1

[59] Heng Li. Minimap2: Pairwise alignment for nucleotide sequences. *Bioinformatics*, 34 (18):3094–3100, 2018. 3.1

[60] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3): 50–60, 2020. 6

[61] Jason Liang, Elliot Meyerson, Babak Hodjat, Dan Fink, Karl Mutch, and Risto Miikkulainen. Evolutionary neural automl for deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 401–409, 2019. 1.1.2

[62] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*, 2019. 1.1.1

[63] Shih-Wei Lin, Kuo-Ching Ying, Shih-Chieh Chen, and Zne-Jung Lee. Particle swarm optimization for parameter determination and feature selection of support vector machines.

Expert Systems with Applications, 35(4):1817–1824, 2008. 1.1.2

[64] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 4.3.3, 4.3.4

[65] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 481–488, 2017. 1.1.2

[66] X. Lu, J. Gonzalez, Z. Dai, and N. Lawrence. Structured variationally auto-encoded optimization. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3267–3275, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/lu18c.html`. 1, 1.2, 2.1, 2.3, 2.3, 2.4

[67] Jiancheng Lyu, Shuai Zhang, Yingyong Qi, and Jack Xin. AutoShuffleNet: Learning permutation matrices via an exact Lipschitz continuous penalty in deep convolutional neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 608–616, 2020. 5.3.3

[68] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 5.5

[69] G. Malkomes, C. Schaff, and R. Garnett. Bayesian optimization for automated model selection. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2900–2908, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. URL `http://dl.acm.org/citation.cfm?id=3157382.3157422`. 1, 1.2, 2.1, 2.3

[70] Guillaume Marçais, David Pellow, Daniel Bork, et al. Improving the performance of

minimizers and winnowing schemes. *Bioinformatics*, 33(14):i110–i117, 2017. 1.2, 1.3, 3.1, 5, 3.2.3, 3.3.1, 3.3.2, 3.4

[71] Guillaume Marçais, Dan DeBlasio, and Carl Kingsford. Asymptotically optimal minimizers schemes. *Bioinformatics*, 34(13):i13–i22, 2018. 3.2.3, 6

[72] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. AISTATS*, pages 1273–1282, 2017. 1.3, 4.1, 4.3.4, 4.4.2

[73] G. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning latent permutations with Gumbel-Sinkhorn networks. In *Proc. ICLR*, 2018. 5.1, 5.1, 5.4, 5.5

[74] Karen H. Miga, Sergey Koren, Arang Rhie, et al. Telomere-to-telomere assembly of a complete human X chromosome. *Nature*, 585(7823):79–84, 2020. 3.3.4, 3.1

[75] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019. 1.1.2, 4.1

[76] Risto Miikkulainen, Olivier Francon, Elliot Meyerson, Xin Qiu, Darren Sargent, Elisa Canzani, and Babak Hodjat. From prediction to prescription: evolutionary optimization of nonpharmaceutical interventions in the COVID-19 pandemic. *IEEE Transactions on Evolutionary Computation*, 25(2):386–401, 2021. 1.1.2

[77] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017. 2.3.4

[78] Randal S Olson and Jason H Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on Automatic Machine Learning*, pages 66–74. PMLR, 2016. 1.1.2

[79] Yaron Orenstein, David Pellow, Guillaume Marçais, et al. Designing small universal

k-mer hitting sets for improved analysis of high-throughput sequencing. *PLOS Computational Biology*, 13:e1005777, 10 2017. 3.2.3

[80] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 1, 1.2, 4.1, 4.3.1

[81] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in Neural Information Processing Systems*, 32, 2019. 5.1, 5.3.1

[82] P. S. Rana. Physicochemical Properties of Protein Tertiary Structure Data Set, 2013. URL `http://archive.ics.uci.edu/ml/datasets/`. 2.4, 2.4.2

[83] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. 1, 1.1.3, 2.2, 2.3.3, 2.5, 6

[84] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations (ICLR)*, 2017. 5.5

[85] John R Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976. 1

[86] Michael Roberts, Wayne Hayes, Brian Hunt, et al. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20:3363–9, 01 2005. 3.1

[87] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. 5.5

[88] Kristoffer Sahlin. Strobealign: flexible seed size enables ultra-fast and accurate read alignment. *Genome Biology*, 23(1):260, 2022. 6

[89] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms

for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 76–85, 2003. 1.3, 3.1, 3.2.1, 3.4

[90] Jim Shaw and Yun William Yu. Theory of local k-mer selection with applications to long-read alignment. *Bioinformatics*, pages 4659–4669, 2021. ISSN 1367-4803. 1.3, 3.1, 3.2.2, 3.4

[91] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. 5.3.3

[92] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in Neural Information Processing Systems*, 30, 2017. 5.3.1, 5.4, 5.2, 5.4, 5.4, 5.5

[93] J. Snoek, L. Hugo, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. NIPS*, pages 2960–2968, 2012. 1.1.3, 2.1, 2.3.3

[94] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. ICML*, pages 1015–1022, 2010. 1.1.3

[95] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *NeurIPS*, pages 2440–2448, 2015. 5.3.1

[96] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013. 1

[97] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples.

In *International Conference on Learning Representations*, 2020. 5.5

[98] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017. 4.1

[99] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 29, 2016. 5.3.1, 5.5

[100] Risto Vuorio, Shao-Hua Sun, Hexiang Hu, and Joseph J Lim. Multimodal model-agnostic meta-learning via task-aware modulation. *Advances in Neural Information Processing Systems*, 32, 2019. 5.1, 5.3.2, 5.4, 5.4, 5.5

[101] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas. Bayesian optimization in a billion dimensions via random embeddings. *JAIR*, 55:361–387, 2016. 2.4, 2.4

[102] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378. PMLR, 2016. 6, 6

[103] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1): 3–34, 2015. 1.1.1

[104] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018. 4.1, 4.1, 4.3.3, 4.5

[105] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *International Conference on Machine Learning*, pages 7045–7054. PMLR, 2019. 5.1, 5.3.2, 5.4, 5.4

[106] Chengxi Ye, Zhanshan Sam Ma, Charles H. Cannon, et al. Exploiting sparseness in de novo genome assembly. In *BMC Bioinformatics*, volume 13, pages 1–8. BioMed Central, 2012. 3.1

[107] J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn. Baysian model-agnostic meta-

learning. In *Proc. NeurIPS*, 2018. 5.1, 5.3.1

[108] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. 5.1, 5.3.3

[109] Zhengdong D. Zhang, Alberto Paccanaro, Yutao Fu, et al. Statistical analysis of the genomic distribution and correlation of regulatory elements in the encode regions. *Genome Research*, 17(6):787–797, 2007. 3.2.3

[110] Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Improved design and analysis of practical minimizers. *Bioinformatics*, 36(Supplement_1):i119–i127, 2020. 1, 1.2, 3.2.3, 3.3, 3.3.4, 3.3.4, 3.3.4, 3.4, 3.4.2, 3.4.5, 3.4.5, 3.4.5

[111] Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Sequence-specific minimizers via polar sets. *Bioinformatics*, 37:i187–i195, 2021. 1, 1.2, 3.1, 3.2.3, 3.3, 3.3.4, 3.3.4, 3.4

[112] P. Zhou, Y. Zou, X. Yuan J. Feng, C. Xiong, and S. Hoi. Task similarity aware meta learning: Theory-inspired improvement on MAML. In *Uncertainty in Artificial Intelligence*, pages 23–33. PMLR, 2021. 5.1, 5.3.2

[113] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 1.2, 4.1, 4.1