University of California
Santa Barbara

Compressed Training for Uncertainty-Aware Compact Neural Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Mathematics

by

Cole P. Hawkins

Committee in charge:

Professor Zheng Zhang, Chair
Professor Paul Atzberger
Professor Peng Li
Dr. Xing Liu

March 2022

The Dissertation of Cole P. Hawkins is approved.

Professor Paul Atzberger

Professor Peng Li

Dr. Xing Liu

Professor Zheng Zhang, Committee Chair

February 2022

Compressed Training for Uncertainty-Aware Compact Neural Networks

by

Cole P. Hawkins

To my parents

Acknowledgements

Thank you to my advisor for his encouragement, support and guidance during my entire grad school experience. He took me on as a student when I didn't know anything, and helped me grow into an independent researcher. The time, energy, and commitment that he put into my development gave me the confidence and skills that I needed to see this through.

Thank you to everyone that I worked with both at UCSB and in industry, with a special thanks to my committee members. You made work a warm and friendly place where I got the chance to learn from so many kind, talented, and generous people. That's been priceless.

Thank you to my friends, who made my time in Santa Barbara wonderful.

Thank you to my family for their unconditional love and support. Without it, none of this would have been possible.

Curriculum Vitæ

Cole P. Hawkins

Education

2022	Ph.D. in Mathematics (Expected), University of California, Santa Barbara.
2016	B.A. in Mathematics, Amherst College

Publications and Articles

1. Hawkins, C.; Koppel, A., Zhang, Z. Online, Informative MCMC Thinning with Kernelized Stein Discrepancy. Under submission. arXiv:2201.07130.
2. Zhang, K.; Hawkins, C.; Zhang, Z. General-Purpose Bayesian Tensor Learning With Automatic Rank Determination and Uncertainty Quantification. Frontiers in Artificial Intelligence 2022, 4.
3. Hawkins, C.; Yang, H.; Li, M.; Lai, L.; Chandra, V. Low-Rank+Sparse Tensor Compression for Neural Networks. arXiv:2111.01697 [cs] 2021.
4. Hawkins, C.; Ioannidis, V. N.; Adeshina, S.; Karypis, G. Scalable Consistency Training for Graph Neural Networks via Self-Ensemble Self-Distillation. AAAI 2022 Workshop: Deep Learning on Graphs, Methods and Applications.
5. Chen, Y.; Hawkins, C.; Zhang, K.; Zhang, Z.; Hao, C. 3U-EdgeAI: Ultra-Low Memory Training, Ultra-Low Bitwidth Quantization, and Ultra-Low Latency Acceleration. GLSVLSI '21: Proceedings of the 2021 on Great Lakes Symposium on VLSI.
6. Zhang, K.; Hawkins, C.; Zhang, X.; Hao, C.; Zhang, Z. On-FPGA Training with Ultra Memory Reduction: A Low-Precision Tensor Method. ICLR Workshop on Hardware-Aware Efficient Training (HAET), 2021.
7. Hawkins, C.; Liu, X.; Zhang, Z. Towards Compact Neural Networks via End-to-End Training: A Bayesian Tensor Approach with Automatic Rank Determination. SIAM Journal on Mathematics of Data Science 2022, 46–71. https://doi.org/10.1137/21M1391444.
8. Cui, C.; Hawkins, C.; Zhang, Z. Tensor Methods for Generating Compact Uncertainty Quantification and Deep Learning Models. In 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD); 2019; pp 1–6.
9. Hawkins, C.; Zhang, Z. Bayesian Tensorized Neural Networks with Automatic Rank Selection. Neurocomputing, 2021.
10. Hawkins, C; Zhang Z. Variational Bayesian Inference for Robust Streaming Tensor Factorization and Completion. In 2018 IEEE International Conference on Data Mining (ICDM); 2018; pp 1446–1451.
11. Annunziata, M. T.; Gibbons, C. R.; Hawkins, C.; Sutherland, A. J. Rational Combinations of Betti Diagrams of Complete Intersections. Journal of Algebra and Applications. 2018, 17 (05), 1850079.

Internship Experience

Facebook Reality Labs *Research Scientist Intern* July - September 2021

- Tensor decomposition for compact AR/VR embedded device neural networks.
- MobileNetv3 compression up to 2× with minimal accuracy degradation.

Amazon Web Services *Applied Scientist Intern* March - June 2021

- Developed new algorithm for scalable semi-supervised graph neural network training (submitted ICDM).
- Studied graph neural network ensembling and distillation.

Intel Parallel Computing Lab *Research Scientist Intern* June - Dec 2019

- Collaborated on fast hardware-friendly algorithms for tensorized neural network training.
- Developed streaming tensor factorization algorithms for inference on streaming graphs.

Draftkings *Business Data Analyst Intern* June - Aug 2017

- Deployed R models to predict new customer value for real-time adjustment to online ad spending.
- Used SQL database to build new analytics tables and acquire data for parameter selection.

Abstract

Compressed Training for Uncertainty-Aware Compact Neural Networks

by

Cole P. Hawkins

The rising computational and memory demands of machine learning models, particularly in resource-constrained edge-device settings, motivate us to develop compressed models that reduce both training and inference costs. In this dissertation we develop several algorithms that advance the compressed training capabilities of low-rank tensor models and the uncertainty quantification capabilities of general compressed nonparametric models. First we introduce compressed low-rank tensorized training with rank reduction for the Tensor-Train format. This enables compressed tensorized training with further compression during the training phase. Then we improve our approach in two ways. First we improve scalability by introducing a variational inference algorithm tailored to the tensor learning problem and demonstrate its effectiveness compressing larger-scale models. Second we improve the generality of our approach by extending our tensorized training with rank reduction to all popular low-rank tensor formats. Finally, we introduce a general-purpose algorithm for compressed nonparametric Bayesian learning that can automatically determine the appropriate complexity of nonparametric distributional representations. This algorithm improves the computation/accuracy and storage/accuracy Pareto frontiers over state-of-the-art Bayesian learning approaches.

Contents

Chapter 1

Motivation and Overview

1.1 Motivation

The high computation and memory costs of deploying machine learning models, especially on resource-constrained devices, are the main motivations of the work in this thesis.

1.1.1 Growing Computational Demand

Machine learning models are deployed in an ever-growing number of use cases [1, 2, 3], consuming increasing amounts of compute at a rate exceeding Moore's law scaling (see Figure 1.1). Earl state-of-the-art image recognition models such as LeNet [4] or VGG [5] and Natural Language Processing models such as RNNs [6] required comparatively little compute. State-of-the-art ImageNet architectures between 2010 and 2015 such as AlexNet and VGG required between 10^3 and 10^4 training PetaFLOPs due to the fact that they were shallow, and trained only on supervised data. Over the last decade, constraints on model depth have lessened [1] and semi/self-supervised learning has increased in popularity [2, 7]. The main constraint on model depth was the vanishing gradient problem [8] which made training deep nets difficult. Recent research progress has improved initialization, improved activation functions, and introduced residual connections making deeper and more complex networks easier to train [1,9,10]. Another contributing factor to increased training costs is the increase in dataset size. The rise of large-scale unsupervised pre-training [7] enormously expanded training dataset size, and therefore

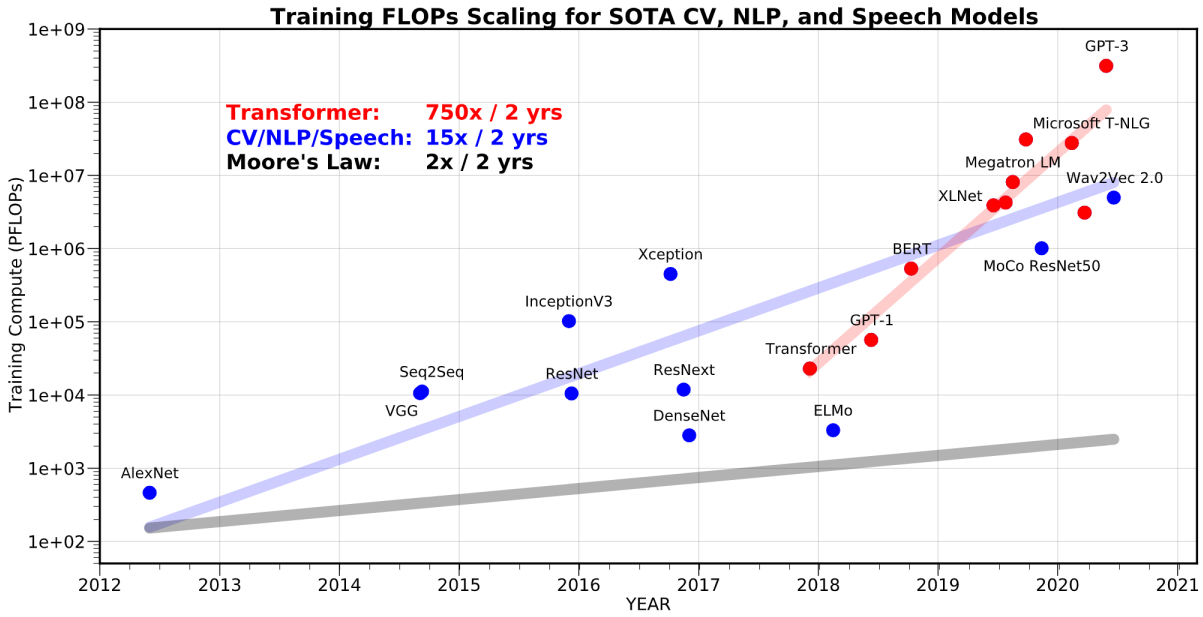


Figure 1.1: Machine learning models demand increasing compute at a rate exceeding Moore's law [13].

increased the required compute. The combination of increased ability to train far deeper and larger models, coupled with growing dataset size, has led to computational demands outpacing Moore's law as state-of-the-art architectures now require upwards of 10^8 PetaFLOPs, a $10,000\times$ increase in the last five years. A specific motivating example is the growing adoption of high-compute Transformer models [2] across many domains [11, 12].

1.1.2 Memory Bottlenecks

Memory capacity and bandwith requirements are also a rising challenge. The larger and deeper models enabled by improved activation functions, improved initialization, and residual connections [1,9,10] naturally have larger parameter counts than shallow models. Another rising trend is the rise of web-scale deployment in Natural Language Processing, Recommendation, and Information Retrieval Systems which require enormous embedding tables [3, 14]. Both larger models and large embedding table requirements have lead to an explosion in parameter counts, illustrated in Figure 1.2. As model memory requirements exceed on-device storage the

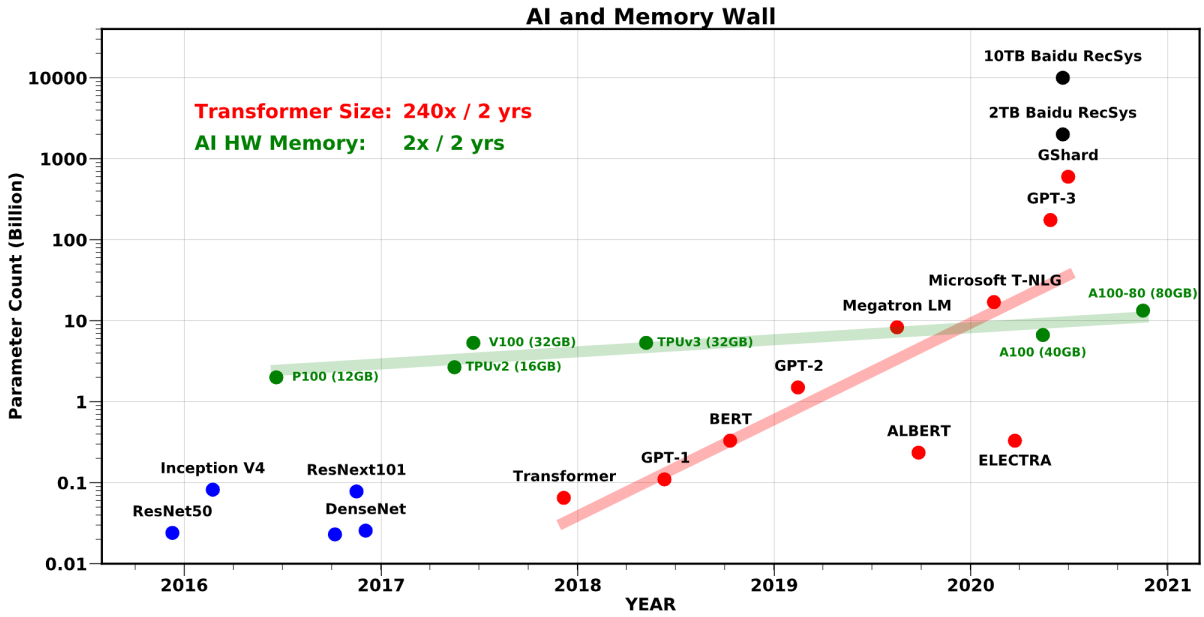


Figure 1.2: Machine learning model memory requirements increase faster than accelerator memory [13].

most pressing challenge is not just memory capacity, but memory bandwith.

Increasing FLOPs requirements have spurred the development of custom accelerators to match the growing need [15,16]. In contrast, memory bandwith used to transfer both data and model parameters has increased at a much slower rate (see Figure 1.3). Not only is memory bandwith a latency limitation but DRAM reads often dominate energy costs of inference and training especially on resource-constrained devices (IoT, mobile phones) where on-chip memory is limited [17,18,19,20]. Reading from DRAM and SRAM can incur energy costs 10× to 100× more than low-precision multiply/accumulate (MAC) operations (see [20] Figure 8).

1.1.3 On-Device Intelligence

FLOPs and memory bandwith requirements apply in all settings in which machine learning models are deployed but these challenges are far more pronounced on edge devices. Edge devices are limited by size and do not have continuous access to a power source. These size constraints limit on-chip memory for accelerators, further increasing power requirements of large

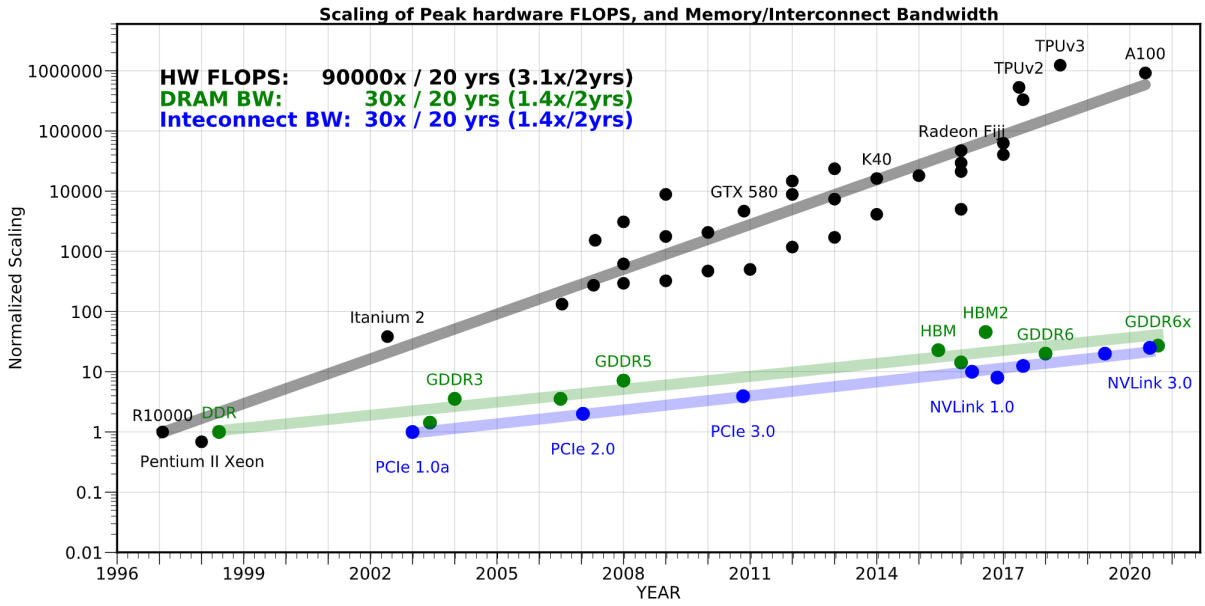


Figure 1.3: FLOPs capabilities of custom machine learning accelerator hardware grows at a faster rate than system memory bandwith [13].

models due to off-chip memory reads [17, 21] in a setting where power usage is a major limiting factor. If resource-constrained IoT devices are inference-only and model training takes place on cloud providers, this reduces training costs. However growing data privacy concerns [22] and continual on-device learning [23] favor that training takes place on resource-constrained edge devices. A final requirement for on-device intelligence to operate safely "in-the-wild" is uncertainty quantification to avoid actions with unknown or potentially unsafe consequences. This additional desire for uncertainty quantification further increases model storage and compute requirements as model ensembles provide state-of-the-art performance [24, 25, 26]. The methods proposed in this dissertation are a step towards enabling the training of compact and uncertainty-aware models on devices with limited memory, power, and compute.

1.2 Related Work

In this section we survey related work in two broad areas: model compression (pruning, quantization, low-rank compression, distillation) and Bayesian deep learning. Many of these

approaches are orthogonal, and can be combined to further improve compression. In each area we discuss the state-of-the-art approach and present any limitations. Finally, we describe the contributions of this thesis in the context of these related works.

1.2.1 Sparse Pruning

Sparse pruning reduces storage costs by setting a subset of weights to 0. We prune a tensor weight $\boldsymbol{\mathcal{A}}$ by applying a binary tensor mask $\boldsymbol{\mathcal{M}}$ to $\boldsymbol{\mathcal{A}}$ using the entry-wise product

$$(\boldsymbol{\mathcal{M}} \odot \boldsymbol{\mathcal{A}})_{i_1 \dots i_d} = a_{i_1 \dots i_d} \times a_{i_1 \dots i_d}.$$

In practice, this means that one can store $\boldsymbol{\mathcal{S}}$ in sparse format. We note that storage costs for sparse matrices exceed the number of nonzero entries due to indexing overhead. Popular indexing formats include bitmap, Compressed Sparse Row/Column (CSR/CSC) for matrices, and Coordinate (COO) format for tensors. The best storage and indexing format depends on the sparsity level [27]. The most popular and generally applicable technique is global magnitude pruning [28,29,30] which zeroes the bottom $100 \times (s_t - s_{t-1})$ percent of the non-masked weights (sorted by l_1 norm) in $\boldsymbol{\mathcal{A}}$ by setting their corresponding entries in $\boldsymbol{\mathcal{M}}$ to zero where s_t is the desired sparsity level at time step t. A popular step schedule is the cubic function introduced by [31]

$$s_t = s_f \left(\frac{t}{n} \right)^3 \tag{1.1}$$

where s_f is the desired final sparsity ratio, t is the current step, and n is the total number of steps. This process gradually increases the sparsity in $\boldsymbol{\mathcal{M}}$ until the target sparsity s_f is reached. An open question is whether to prune entries during the training phase or after. An enormous body of literature proposes a wide range of gradient and magnitude-based techniques, however simple magnitude-based approaches outperform most methods and have high generality [30]. Pruning can either be unstructured, in which all weights are pruned independently, or structured, and weights are pruned in groups. The advantage of structured pruning is that

it yields more computationally friendly (often dense) models [27] by pruning groups such as matrix columns or convolutional channels. The advantage of unstructured pruning is that it usually leads to greater memory cost reduction at the expense of higher inference costs [30].

After Training In this setting a large fully-trained network is iteratively pruned and fine-tuned on the training data [31]. The fine-tuning loss can either be the original training loss (i.e. cross-entropy) or a distillation loss (see (1.4)) where the unpruned model serves as the teacher. This approach incurs the high cost of uncompressed training, and unstructured post-training pruning may incur high practical energy costs during inference unless the hardware and pruning approaches are tightly coupled due to the irregular computation and memory access patterns introduced by sparsity [17, 21, 27].

During Training It is possible to prune neural networks before training and accelerate the entire training process [32, 33] but this is not a mature research area. The Lottery Ticket Hypothesis [34] and its variants [35, 36] are the main approach in this area. During training the network is iteratively pruned using a pruning schedule such as (1.1). Often, at each pruning stage the nonzero weights are "rewound" by resetting the nonzero weights to their values at an earlier stage of training. The advantage of pruning during training is the reduction of both training and inference costs [32, 33] but the challenge is pre-specification of the pruning schedule and desired compression ratio.

1.2.2 Quantization

Quantization lowers the bitwidth in neural networks to reduce memory and computational requirements. The standard bitwidth is 32-bit floating point (FP32) and neural network weights, activations, and gradients can all be quantized to a lower bitwidths such as binary (1-bit) or INT4/8 (4/8-bit). Low bitwidths are particularly well-suited to low-resource devices. The literature on quantization is vast, so we refer the reader to [37] for a thorough survey. We review work on weight quantization which is most related to our proposed compression methods. Let

x be a single (full-precision) scalar weight in a neural network. The basic uniform quantization operation [37] is

$$Q(x) = S \cdot \texttt{round}\left(\frac{x}{S}\right) \tag{1.2}$$

where S is the quantization step length. As an example, when x is bounded between 0 and 1 we can perform binary quantization with $S = 1$ and the rounding function

$$\texttt{round}(z) = \begin{cases} 1, \ z > 0.5 \\ \\ 0, \ z \leq 0.5 \end{cases} . \tag{1.3}$$

More complicated rounding operations are required for non-uniform quantization schemes in which S varies based on the quantization level and becomes a function of x. We note that quantization schemes can be pre-fixed or may involve learned parameters.

After Training Work in [29] renewed interest in quantization by combining pruning, clustering, and quantization to drastically reduce memory costs. Given a pre-trained network, one can quantize the weights by testing various bitwidths on the validation set to establish a compression/accuracy Pareto frontier. More complicated bitwidth selection and rounding schemes are challenging open problems [38, 39]. Post-training quantization suffers from the same drawbacks as other compression methods: no efficiency gains are produced during the training phase, and the final results may be lower accuracy [40]. The advantage of post-training quantization is simplicity, and the disadvantages are accuracy loss and higher training costs.

Before Training Instead of the train-then-compress approach, more recent work focuses on training in low precision [38, 40, 41]. The weights are stored at a low bitwidth during the entire training process, reducing memory and compute cost. The benefits of this approach are clear as memory requirements depend linearly on the bitwidth and computation costs are often. Gradient and activation quantization are particular challenges during training. Especially at low bitwidth, quantization can introduce signficant errors into the optimization process and

render operations non-differentiable. A popular approach to deal with the non-differentiability challenge is the straight-through estimator [42]. Learned quantization schemes can be used to reduce quantization error [37, 40]. Finally, full-precision gradients can be used to stabilize training [39]. We note the highly related work of quantized tensorized training with rank determination which appeared in [18] and is an FPGA-acclerated, low-precision extension of methods proposed in Chapter 3.

1.2.3 Distillation

The goal of knowledge distillation is to transfer the knowledge of an expensive teacher model to a cheaper student model [43]. In this setting "expensive" refers to high memory or computation costs. The expensive teacher model is f_t, the "cheap" student model is f_s. Given a training example with input $\mathbf{x}$ and label $\mathbf{y}$ the standard knowledge distillation loss is

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \alpha l(\mathbf{x}, \mathbf{y}) + (1 - \alpha) T^2 KL\left(f_s(\mathbf{x})/T || f_t(\mathbf{x})/T\right) \tag{1.4}$$

where l is any standard loss function and KL is the KL-divergence. In the classification setting l is the cross-entropy loss. The KL term in the loss function penalizes the student output distribution when it differs from the teacher output distribution. Both outputs are temperature scaled by T, and a common setting is $\alpha = 0.9$ and $T = 3$ [43, 44]. Higher values of α increase the weight of the standard training loss. Higher values of T encourage the student network to match the teacher on predictions outside of the top-1 predicted class.

Distillation usually requires high training costs because the expensive teacher must be trained before the student is trained with the distillation loss. Also, distillation requires approximately $2\times$ the inference computation as standard training because the teacher predictions must be computed at each step. Therefore distillation decreases inference costs but not training costs.

1.2.4 Low-Rank Compression

The goal of low-rank compression in tensor or matrix format is to reduce both the memory and compute costs. Neural network parameters (i.e. FC layers, convolutional kernels) are replaced by low-rank matrix or tensor factors. This compression format can be applied post-training via factorization, or pre-training and then trained end-to-end.

After Training In post-training compression, a full uncompressed model is trained to completion. Then the model weights can be approximated using low-rank decomposition to reduce storage costs and FLOPS during the inference stage. Often the model is fine-tuned end-to-end after applying low-rank decomposition. Early work on post-training compression focused on low-rank matrix format for both FC layers and convolutional kernels [45, 46]. Soon after, both [45, 47] considered the low-rank CP format to reshape and then compress both FC layers and convolutional kernels. The Tucker decomposition was also considered in [48]. Matrix compression methods (i.e. SVD) can be applied to either matrices in FC layers or reshaped convolutional kernels. Similarly, low-rank tensor compression methods (i.e. CP, Tucker, TT, TTM) can be directly applied to convolutional kernels or to reshaped matrices from FC layers (see Section 2.2).

The advantage of post-training compression is that accurate models can be obtained easily from public repositories. Factorization operations such as the SVD or Alternating Least Squares (ALS) [49] are cheap relative to high training costs. The disadvantage of post-training compression is that it requires expensive full-model training and only reduces compute and memory requirements during the inference stage. When model training must be performed on edge devices, full-model training costs may be prohibitive.

Before Training An alternative to post-training compression is end-to-end low-rank training [50]. In this approach the neural network factors are expressed in low-rank format during the entire training process. This approach was investigated for FC layers in TTM format by [50], for both FC and convolutional layers by tensor train decomposition in [51], and by Tucker

decomposition in [52]. Work in [48,50] demonstrates that tensor-compression can reduce storage cost, inference and training times, and FLOP count.

The main advantage of pre-training compression is that these performance gains apply during both training and inference. The main challenge of compressed training is pre-specifying the tensor rank parameter. Existing work specifies the tensor rank parameter (and therefore the compression ratio) in advance [50,51,52]. Setting the rank too high reduces the compression ratio, but setting the rank too low may impact model accuracy. This makes the rank parameter difficult to determine without expensive hyperparameter optimization schemes. However performing many training runs for parameter selection destroys the gains of compressed training, rendering grid search an unsuitable choice for edge devices. The problem of selecting a tensor rank $d$uring end-to-end compressed training is a major motivating problem of this work.

Rank Determination Rank determination is a major challenge for compression via low-rank tensor and matrix methods. In the post-training compression approach the rank can be determined using SVD eigenvalues or repeated parameter decomposition to explore the fidelity/compression frontier. However this incurs the high cost of uncompressed training. A key challenge is therefore to determine the tensor rank during neural network training.

In practice determining a proper tensor rank *a priori* is hard, and a bad rank estimation can result in low accuracy or high training cost. This challenge is the main motivation of our work. Tensor rank determination is heavily studied in the tensor completion problem. Two main approaches are used for rank determination in tensor completion: low-rank optimization and Bayesian inference. Optimization methods mainly rely on some generalization of the matrix nuclear norm [53] to tensors. The most popular approaches place a low-rank objective on tensor unfoldings [54, 55, 56], but the computation is expensive for high-order tensors. The Frobenius norms of CP factors are used as a regularization for 3-way tensor completion [57], but this does not generalize to high-order tensors either. Bayesian methods can directly infer the tensor rank in CP or Tucker tensor completion through low-rank priors [57, 58, 59, 60, 61]. In the CP and Tucker formats, the ranks of different tensor factors do not couple with each other. This is not

true in the tensor-train format (see Section 3.1.2).

The key advantage of Bayesian methods is that an appropriate low-rank prior allows one-shot tensor completion with rank determination, and no grid search to determine nuclear norm regularization weights or tensor ranks. Existing Bayesian tensor methods for parameter inference and rank determination are only suitable for linear tensor problems [57, 58, 62]. Current Bayesian tensor methods solve tensor factorization, completion and regression problems on small-scale data where the observed data is a linear function of the hidden tensor [58, 60, 63], whereas the mapping is nonlinear in neural networks. Therefore, previous work using mean-field variational inference cannot generalize to tensorized neural networks. However they provide a strong foundation on which to build rank determination methods for nonlinear tensor problems such as tensor rank determination in tensorized neural networks.

1.2.5 Bayesian Neural Networks

Bayesian neural networks model a distribution over neural network model parameters, and therefore implicitly predictive distributions. The two primary reasons that practitioners employ Bayesian techniques in neural network training are:

- **Increased accuracy** through Bayesian model averaging [64, 65, 66] and
- **Calibrated uncertainty estimates** which accurately estimate the confidence of the model prediction [67, 68].

The main challenge in Bayesian learning for deep neural networks is the tradeoff between prediction quality (calibration and accuracy) and model complexity. The technical goal is to learn a distribution q that approximates a target distribution over the model weights p as in Section 2.3. Bayesian inference approaches are often limited by model storage and inference costs as they have higher requirements than deterministic neural networks. We cover the technical details of several nonparametric and parametric approaches in Section 2.3. Here we focus their advantages and limitations. We will often refer to the complexity/consistency

tradeoff. The model complexity is the cost of storage and inference, and the consistency is the fidelity of the representation of the target posterior distribution p. Work in [68] links higher consistency representations to higher accuracy and better calibration [68].

Nonparametric Sampling-Based Approaches Nonparametric approaches represent q as a set of particles and therefore are flexible and can provide high-quality representations as the number of particles grows. High-quality nonparametric representations consisting of many particles may lead to well-calibrated predictions and high accuracy, but can be prohibitively expensive since training storage costs and inference storage/compute costs are high. Representations consisting of as many as $1,000$ particles [68] in state-of-the-art Hamiltonian Monte Carlo approaches are not feasible for real-time uncertainty-aware decision making. Therefore a major challenge in nonparametric learning is to improve the complexity/consistency tradeoff by compressing the approximating distribution q during both the training and inference process.

Parametric Inference Approaches Most parametric Bayesian inference methods can be formulated as variants of Stochastic Variational Inference [25, 69]. Parametric variational inference-based methods operate at the low-complexity end of the complexity/consistency tradeoff. They are cheap to train, with similar computational requirements to deterministic neural networks [70]. However their predictive estimates, obtained by repeatedly sampling the learned weight distribution during inference, underperform nonparametric approaches as compute and storage budgets grow [68].

1.3 Overview and Contributions

Despite their success in many applications, deep neural networks are often over-parameterized, requiring extensive computing resources in their training and inference. For instance, the VGG-19 network requires 500M memory [5] for image recognition and realistic Deep Learning Recommendation Models (DLRM) [3] have billions of parameters. It has been a common practice

to reduce the size of neural networks before deploying them in various scenarios ranging from cloud services to embedded systems to mobile applications. To reduce hardware cost, numerous techniques have been developed to build *compact* models [28, 71, 72] after training. Representative approaches include pruning [28, 73], quantization [29, 74, 75], knowledge distillation [43], and low-rank factorization [47, 48, 76, 77, 78]. Among these techniques, low-rank tensor compression [47, 48, 51, 79, 80, 81] has achieved possibly the most significant compression, leading to promising reduction of FLOPS and memory requirements [19, 48]. The recent progress of algorithm/hardware co-design [19, 82] of tensor operations can further reduce the run-time and boost the energy efficiency of tensorized models on edge devices (e.g., on FPGA and ASIC). While post-training compression techniques can reduce the cost of deploying a deep neural network, they cannot reduce the training cost. Pruning techniques can also be used in training [73, 83], they do not necessarily reduce the number of training variables or decrease the energy costs. Low-precision arithmetic [38, 84, 85, 86] can reduce the cost per parameter during training and inference, but quantization decreases training stability and the memory cost reduction is limited to a single order of magnitude even in ultra low-precision 4-bit training [86].

Low-rank tensor compression for neural networks presents the opportunity for compressed training. However existing methods train tensorized neural networks by pre-specifying the tensor rank, which controls the compression factor [50, 51, 52]. Pre-training rank specification limits the ability of the data to determine the compression tradeoff and, as we show in Chapter 4, can reduce either memory/compute savings or accuracy. Therefore our goal is to train compressed uncertainty-aware neural networks.

1.3.1 Overview

First, in Chapter 2 we present the necessary preliminaries on low-rank tensor compression and Bayesian inference. Next, in Chapter 3 we address

- **Challenge 1**: compressed rank-adaptive training for compressed neural networks

with

- **Contribution 1:** the first rank-adaptive tensorized neural network training scheme. We develop a Bayesian model for the Tensor-Train decomposition [87] that reduces the rank of a low-rank tensor neural network parameter during the training process. We employ Stein Variational Gradient Descent [88] as a flexible nonparametric Bayesian inference method to achieve both rank reduction and uncertainty quantification. This permits compressed tensorized training with rank reduction. Therefore compressed tensorized models can be trained once to achieve strong compression instead of sweeping over possible tensor rank parameters. This contribution lets us train compact models and achieve further compression (via tensor rank reduction) *during compressed training.*

Chapter 3 presents a solution for compressed rank-adaptive training, but raises new challenges.

- **Challenge 2**: Rank-adaptive training is slow compared to standard optimization.
- **Challenge 3**: Chapter 3 only proposes a formulation for one low-rank tensor format.
- **Challenge 4**: The user must specify a model complexity parameter (number of particles) of the Stein Variational Gradient Descent Bayesian inference algorithm.

In Chapter 4 we directly address Challenges 2 and 3 with the following remedies:

- **Contribution 2**: We introduce a scalable variational Bayesian inference solver. The proposed variational inference solver is based on Stochastic Variational Inference [69] and only requires one copy of the *compressed* model parameters for tensorized training with rank reduction via *maximum a posteriori* (MAP) training. To control stochastic gradient variance during the rank reduction process we introduce a specialized EM-style algorithm with closed-form updates. To enable both rank reduction and uncertainty quantification requires only two copies of the *compressed* model parameters.
- **Contribution 3**: We extend our automatic rank determination framework to all low-rank tensor formats in common use (CP, TT, TTM, and Tucker) and demonstrate its

effectiveness on fully-connected layers, convolutions, and embedding tables. This extension enables general-purpose tensor-compressed training with rank reduction by removing tensor format constraints. This is particularly important because different low-rank tensor formats may be better suited to different layer types [48, 51, 89].

The work in Chapter 4 indirectly solves Challenge 4 by using an optimization-based parametric Bayesian inference algorithm, instead of a sampling-based nonparametric Bayesian inference algorithm. The compressed training methods presented in Chapter 4 have been implemented on FPGA where they achieved 59× speedup and 123× energy reduction and 292× memory reduction compared to uncompressed training on embedded CPU [90]. Finally, in Chapter 5 we directly address Challenge 4:

- **Contribution 4**: We introduce an online pruning method to automatically select nonparametric model complexity for sampling-based Bayesian inference algorithms. The general-purpose thinning method we introduce removes unnecessary particles during the Bayesian sampling process. Not only does this reduce memory costs during the inference stage. It also permits advanced samplers [91] to directly target compressed representations during the training phase. Further, we provide convergence guarantees for our approach and demonstrate that our online thinning method introduces no additional asymptotic bias in state-of-the-art samplers [92]. Our online thinning method adds online compression to existing samplers but converges in Kernelized Stein Discrepancy [92] at the same asymptotic rate.

1.3.2 Relation to existing work

In Figure 1.4 we give a high-level visual overview of related research areas and place the work in this dissertation in context. Many methods offer the benefits of compressed training, but only rank-adaptive tensorized training proposed in Chapters 3 and 4 offers the benefits of compressed training and inference while providing further rank reduction during the training phase and uncertainty quantification during the inference phase. The KSD Thinning method

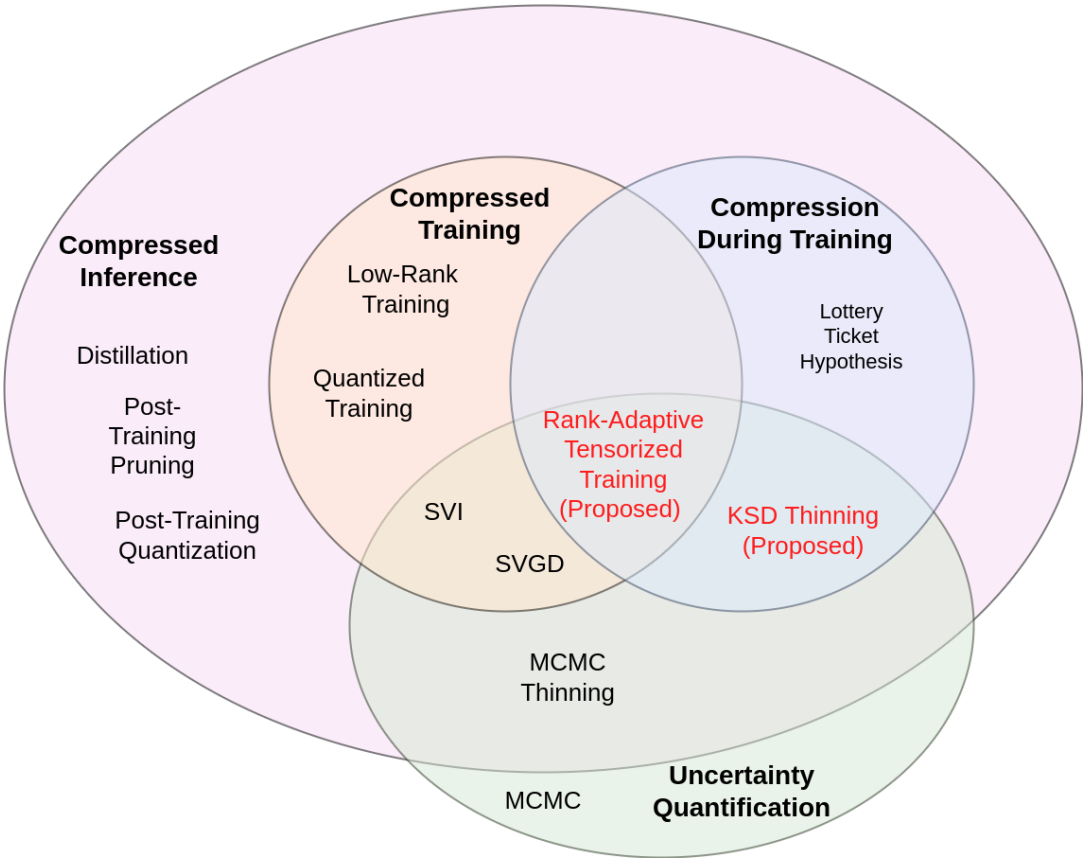


Figure 1.4: High-level overview of the main contributions of this work. Contributions are marked in red. The rank-adaptive tensorized training methods proposed in Chapters 3 and 4 provide uncertainty-aware compressed training with further compression during the training process. The KSD Thinning method proposed in Chapter 5 retains the flexibility of MCMC methods but improves the compression/accuracy tradeoff.

introduced in Chapter 5 improves the complexity/consistency tradeoff between MCMC and Stochastic Variational inference by online thinning of MCMC particles. This approach retains the flexibility of nonparametric methods such as MCMC but improves their particle efficiency.

Chapter 2

Background

2.1 Low-Rank Tensors

This paper uses lower-case letters (e.g., a) to denote scalars, bold lowercase letters (e.g., $\mathbf{a}$) to represent vectors, bold uppercase letters (e.g., $\mathbf{A}$) to represent matrices, and bold calligraphic letters (e.g., $\boldsymbol{\mathcal{A}}$) to denote tensors. A tensor is a generalization of a matrix, or a multi-way data array. An order-d tensor is a d-way data array $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, where I_n is the size of mode n. We use the notation $\boldsymbol{\mathcal{A}}(\dots, i_j, \dots)$ to represent the order $d-1$ subtensor of $\boldsymbol{\mathcal{A}}$ obtained by fixing the j^{th} index to i_j. The $(i_1, i_2, \cdots, i_d)$-th element of $\boldsymbol{\mathcal{A}}$ is denoted as either $a_{i_1 i_2 \cdots i_d}$ or $\boldsymbol{\mathcal{A}}(i_1, \dots, i_d)$. An order-3 tensor is shown in Fig. 2.1 (a).

Definition 2.1.1 *The mode-n product of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_d}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ is*

$$\boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{A}} \times_n \mathbf{U} \Longleftrightarrow b_{i_1 \dots i_{n-1} j i_{n+1} \dots i_d} = \sum_{i_n=1}^{I_n} a_{i_1 \dots i_d} u_{j i_n}. \tag{2.1}$$

The result is still a d-dimensional tensor $\boldsymbol{\mathcal{B}}$, but the mode-n size becomes J. In the special case $J = 1$, the n-th mode diminishes and $\boldsymbol{\mathcal{B}}$ becomes an order-$d-1$ tensor.

A tensor has a massive number of entries if d is large. This causes a high cost in both computing and storage. Fortunately, many practical tensors have a low-rank structure, and this property can be exploited to reduce the cost dramatically.

Definition 2.1.2 *A d-way tensor* $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ *is rank-1 if it can be written as a single outer product of d vectors*

$$\boldsymbol{\mathcal{A}} = \mathbf{u}^{(1)} \circ \cdots \circ \mathbf{u}^{(d)}, \text{ with } \mathbf{u}^{(n)} \in \mathbb{R}^{I_n} \text{ for } n = 1, \cdots, d.$$

Each element of $\boldsymbol{\mathcal{A}}$ is $a_{i_1 i_2 \cdots i_d} = \prod_{n=1}^{d} u_{i_n}^{(n)}$, where $u_{i_n}^{(n)}$ is the i_n-th element of the vector $\mathbf{u}^{(n)}$.

A rank-1 tensor can be stored with only d vectors. Most tensors are not rank-1, but many can be well-approximated via tensor decomposition [49] if their ranks are low. We will use the following four tensor decomposition formats to reduce the parameters of neural networks.

Definition 2.1.3 *The CP factorization [93, 94] expresses tensor* $\boldsymbol{\mathcal{A}}$ *as the sum of multiple rank-1 tensors:*

$$\boldsymbol{\mathcal{A}} = \sum_{j=1}^{R} \mathbf{u}_j^{(1)} \circ \mathbf{u}_j^{(2)} \cdots \circ \mathbf{u}_j^{(d)}. \tag{2.2}$$

Here $\circ$ *denotes an outer product operator. The minimal integer* R *that ensures the equality is called the* **CP rank** *of* $\boldsymbol{\mathcal{A}}$*. To simplify notation we collect the rank-1 terms of the n-th mode into a factor matrix* $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R}$ *with* $\mathbf{U}^{(n)}(:, j) = \mathbf{u}_j^{(n)}$*. A rank-R CP factorization can be described with d factor matrices* $\{\mathbf{U}^{(n)}\}_{n=1}^{d}$ *using* $R \sum_n I_n$ *parameters.*

Definition 2.1.4 *The Tucker factorization [95] expresses a d-way tensor* $\boldsymbol{\mathcal{A}}$ *as a series of mode-n products:*

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}^{(1)} \times_2 \cdots \times_d \mathbf{U}^{(d)}. \tag{2.3}$$

Here $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times \cdots \times R_d}$ *is a small core tensor, and* $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ *is a factor matrix for the n-th mode. The* **Tucker rank** *is the tuple* $(R_1, \ldots, R_d)$*. A Tucker factorization with ranks* $R_n = R$ *requires* $R^d + R \sum_n I_n$ *parameters.*

Definition 2.1.5 *The tensor-train (TT) factorization [87] expresses a d-way tensor* $\boldsymbol{\mathcal{A}}$ *as a collection of matrix products:*

$$a_{i_1 i_2 \ldots i_d} = \boldsymbol{\mathcal{G}}^{(1)}(:, i_1, :) \boldsymbol{\mathcal{G}}^{(2)}(:, i_2, :) \ldots \boldsymbol{\mathcal{G}}^{(d)}(:, i_d, :). \tag{2.4}$$

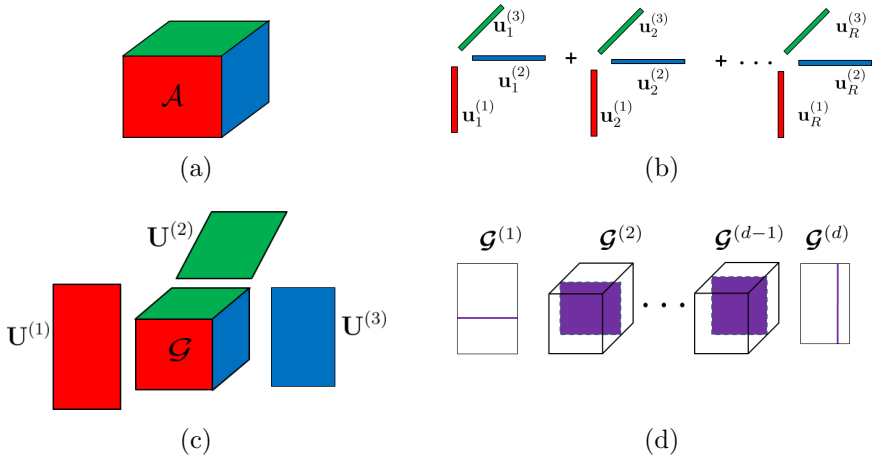


Figure 2.1: (a): An order-3 tensor, (b) and (c): representations in CP and Tucker formats respectively, where low-rank factors are color-coded to indicate the corresponding modes. (d): TT representation of an order-d tensor, where the purple lines and squares indicate $\boldsymbol{\mathcal{G}}^{(n)}(:, i_n, :)$, which is the i_n-th slice of the TT core $\boldsymbol{\mathcal{G}}^{(n)}$ obtained by fixing its second index.

Each TT-core $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ *is an order-3 tensor. The tuple* $(R_0, R_1, \ldots, R_d)$ *is the* **TT-rank** *and* $R_0 = R_d = 1$.

The TT format uses $\sum_n R_{n-1} I_n R_n$ parameters in total and leads to more expressive interactions than the CP format.

Let $\mathbf{A} \in \mathbb{R}^{I \times J}$ be a matrix. We assume that I and J can be factored as follows:

$$I = \prod_{n=1}^{d} I_n, J = \prod_{n=1}^{d} J_n. \tag{2.5}$$

We can reshape $\mathbf{A}$ into a tensor $\boldsymbol{\mathcal{A}}$ with dimensions $I_1 \times \cdots \times I_d \times J_1 \times \cdots \times J_d$, such that the (i, j)-th element of $\mathbf{A}$ uniquely corresonds to the $(i_1, i_2, \cdots, i_d, j_1, j_2, \cdots, j_d)$-th element of $\boldsymbol{\mathcal{A}}$. The TT decomposition can extended to compress the resulting order-$2d$ tensor as follows.

Definition 2.1.6 *The tensor-train matrix (TTM) factorization expresses an order-*$2d$ *tensor* $\boldsymbol{\mathcal{A}}$ *as* d *matrix products:*

$$a_{i_1 \ldots i_d j_1 \ldots j_d} = \boldsymbol{\mathcal{G}}^{(1)}(:, i_1, j_1, :) \boldsymbol{\mathcal{G}}^{(2)}(:, i_2, j_2, :) \ldots \boldsymbol{\mathcal{G}}^{(d)}(:, i_d, j_d, :). \tag{2.6}$$

Each TT-core $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times J_n \times R_n}$ *is an order* 4 *tensor. The tuple* $(R_0, R_1, R_2, \ldots, R_d)$ *is the* **TT-rank** *and as before* $R_0 = R_d = 1$*. This TTM factorization requires* $\sum_n R_{n-1} I_n J_n R_n$ *parameters to represent* $\boldsymbol{\mathcal{A}}$*.*

We provide a visual representation of the CP, Tucker, and TT formats in Fig. 2.1 (b) – (d).

2.2 Tensorized Neural Networks

A deep neural network can be written as

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) = \mathbf{g}_L\left(\mathbf{g}_{L-1}\left(\cdots \mathbf{g}_1(\mathbf{x})\right)\right) \tag{2.7}$$

where $\mathbf{x}$ is an input data sample and $\mathbf{y}$ is an output label. Here $\mathbf{g}_k(\mathbf{z}) = \sigma(\mathbf{W}_k\mathbf{z} + \mathbf{b}_k)$ represents layer k, where σ is a nonlinear activation function, $\mathbf{W}_k$ and $\mathbf{b}_k$ are the weights and bias, respectively. Considering parameter dependence, we can re-write (2.7) as

$$\mathbf{y} = \mathbf{h}(\mathbf{x} \mid \{\mathbf{W}_k, \mathbf{b}_k\}_{k=1}^{L}). \tag{2.8}$$

In a convolutional layer $\mathbf{W}_k$ should be replaced with tensor $\boldsymbol{\mathcal{W}}_k$. In modern neural networks, $\{\mathbf{W}_k\}_{k=1}^{L}$ contain millions to billions of parameters, which cause huge challenges in training and inference on various hardware platforms. A promising solution is to generate a compact neural network via low-rank tensor compression [47, 50, 51] as follows:

- **Folding to high-order tensors.** A weight matrix $\mathbf{W} \in \mathbb{R}^{I \times J}$ can be folded into an order-d tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ where $IJ = \prod_n I_n$. We can also fold $\mathbf{W}$ to an order-$2d$ tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d \times J_1 \times \cdots \times J_d}$ such that $w_{ij} = a_{i_1 \cdots i_d j_1 \cdots j_d}$. While a convolution filter is already a tensor, we can reshape it to a higher-order tensor with reduced mode sizes.

- **Low-rank tensor compression.** After folding $\mathbf{W}$ into a higher-order tensor $\boldsymbol{\mathcal{A}}$, one can employ low-rank tensor compression to reduce the number of parameters. Either the CP, Tucker, TT or TTM factorization can be applied [47, 48, 51].

Assume that $\mathbf{\Phi}_k$ includes all low-rank tensor factors required to represent $\mathbf{W}_k$. Considering the dependence of $\mathbf{W}_k$ on $\mathbf{\Phi}_k$, we can now write (2.8) as

$$\mathbf{y} = \mathbf{h}\left(\mathbf{x} \mid \{\mathbf{W}_k\left(\mathbf{\Phi}_k\right), \mathbf{b}_k\}_{k=1}^{L}\right) = \mathbf{f}\left(\mathbf{x} \mid \mathbf{\Psi}\right), \text{ with } \mathbf{\Psi} = \{\mathbf{\Phi}_k, \mathbf{b}_k\}_{k=1}^{L}. \tag{2.9}$$

Here $\mathbf{\Psi}$ include all tensor factors and bias vectors in a tensorized neural network. The number of variables in $\mathbf{\Psi}$ is often orders-of-magnitude smaller than that in the original model (2.8).

Please note the following:

- The tensor factors in $\mathbf{\Phi}_k$ depend on the tensor format we choose. In CP format, $\mathbf{\Phi}_k$ includes d matrix factors; in Tucker format, $\mathbf{\Phi}_k$ includes d factor matrices and a small order-d core tensor as shown in (2.3); when the TT or TTM format is used, $\mathbf{\Phi}_k$ includes d order-3 or order-4 TT cores shown in (2.4) and (2.6) respectively.

- The number of variables in each $\mathbf{\Phi}_k$ depends on the tensor ranks used in the compression. A higher tensor rank leads to higher expressive power but a lower compression ratio. In existing approaches, it is hard to select a proper tensor rank *a-priori*.

Two main approaches exist to produce low-rank tensorized neural networks. The first approach trains an uncompressed neural network $\mathbf{h}$ and then performs tensor factorization on each of the weights $\{\mathbf{W}_k\}_{k=1}^{L}$. This train-then-compress approach suffers from two drawbacks:

- **High training costs.** The uncompressed training consumes a huge amount of memory, run-time, and energy on a hardware platform.

- **Lower accuracy.** The subsequent tensor compression causes accuracy loss, which becomes significant when the compression ratio is high.

The second approach is fixed-rank tensorized training. In this approach the user pre-specifies the tensor rank and trains low-rank tensor factors of weight parameters. This approach avoids the compute and memory requirements of uncompressed training but requires that the user

manually select a good rank *a-priori*. This approach usually requires multiple training runs to select the rank. In addition a user-specified rank may achieve suboptimal compression.

2.3 Bayesian Inference

The goal of Bayesian inference is to infer the parameters $\boldsymbol{\theta}$ of a posterior distribution

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D})|\boldsymbol{\theta})p(\boldsymbol{\theta}}{p(\mathcal{D})}. \tag{2.10}$$

In most large-scale machine learning settings $p(\mathcal{D})$ is intractable to compute, but we do have access to the gradient of the log-posterior function with respect to $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\mathcal{D}) = \nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}). \tag{2.11}$$

The goal is to generate an approximate representation $q(\boldsymbol{\theta})$ of the posterior distribution $p(\boldsymbol{\theta})$. In the following subsections we present several methods for constructing q.

2.3.1 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods sequentially generate a set of particles $\mathbf{D} = \{\boldsymbol{\theta}^i\}_{i=1}^n$ such that the empirical density $q_{\mathbf{D}}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \delta_{\boldsymbol{\theta}_i}$ approximates the posterior $p(\boldsymbol{\theta}|\mathcal{D})$, where $\delta_{\boldsymbol{\theta}_i}$ is the indicator function at $\boldsymbol{\theta}_i$. The particles are generated by the time steps of a Markov Chain whose target distribution is the true posterior $p(\boldsymbol{\theta}|\mathcal{D})$. There are many ways of constructing a Markov Chain, but we focus on two: Random Walk Metropolis (RWM) and the Metropolis-Adjusted Langevin Algorithm (MALA).

Given the current point $\boldsymbol{\theta}_i$, both RWM and MALA consist of two phases: a proposal step and an accept/reject step. The RWM proposal step generates the proposal $\boldsymbol{\theta}'$ by

$$\boldsymbol{\theta}' \sim \mathcal{N}(\boldsymbol{\theta}_i, tI) \tag{2.12}$$

where t is a tunable scale parameter. The MALA proposal generates the proposal by

$$\boldsymbol{\theta}' \sim \mathcal{N}\left(\boldsymbol{\theta}_i + \frac{t}{2}\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta}_i|\mathcal{D}), tI\right) \tag{2.13}$$

and t is the tunable step size parameter. The posterior gradient term in the MALA proposal often leads to faster mixing compared to methods such as RWM which do not use the posterior gradient.

The second step is the Metropolis-Hastings accept/reject step. This procedure computes a tolerance parameter α based on the posterior probability of the proposal and accepts or rejects the new sample $\boldsymbol{\theta}'$ if $\alpha > u$ where $u \sim U[0, 1]$. To compute the threshold:

$$\alpha = \frac{p(\boldsymbol{\theta}')|\mathcal{D})}{p(\boldsymbol{\theta}_i|\mathcal{D})}. \tag{2.14}$$

Therefore if the new proposed sample $\boldsymbol{\theta}'$ has higher posterior probability than the current sample $\boldsymbol{\theta}_i$ then $\alpha > 1$ and the new sample is automatically accepted and $\boldsymbol{\theta}_i = \boldsymbol{\theta}'$. This accept/reject step ensures that q converges to the correct target density p.

MCMC methods are the gold standard for posterior inference. Given unlimited time, they generate highly accurate representations of the target posterior. However they suffer from two key drawbacks:

- **Sample Complexity:** MCMC methods require a large number of samples (model copies) which leads to high compute and memory costs.
- **Training Time:** MCMC methods require long sampling periods to converge, especially in high-dimensional spaces.

Next we introduce two Bayesian inference methods that address these drawbacks.

2.3.2 Stein Variational Gradient Descent (SVGD)

Stein Variational gradient descent offers a tradeoff between the flexibility of nonparametric MCMC methods with the speed of variational Bayesian inference. The goal is to find a set of particles $\mathbf{D} = \{\boldsymbol{\theta}^i\}_{i=1}^n$ such that $q_{\mathbf{D}}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} k(\boldsymbol{\theta}, \boldsymbol{\theta}^i)$ approximates the true posterior $p(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}})$. Here $k(\cdot,\cdot)$ is a positive definite kernel such as the Radial Basis Function. The particles can be found by minimizing the Kullback–Leibler divergence between $q_{\mathbf{D}}(\boldsymbol{\theta})$ and $p\left(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}}\right)$). The optimal update $\phi(\cdot)$ is derived in [88] and takes the form

$$\begin{aligned} \boldsymbol{\theta}^k &\leftarrow \boldsymbol{\theta}^k + \epsilon\phi(\boldsymbol{\theta}^k) \\ \phi(\boldsymbol{\theta}^k) &= \frac{1}{n}\sum_{i=1}^{n}\Big[k(\boldsymbol{\theta}^i, \boldsymbol{\theta}^k)\nabla_{\boldsymbol{\theta}^i}\log p(\boldsymbol{\theta}^i|\boldsymbol{\mathcal{D}}) + \nabla_{\boldsymbol{\theta}^i} k(\boldsymbol{\theta}^i, \boldsymbol{\theta}^k)\Big] \end{aligned} \tag{2.15}$$

where ϵ is the step size.

The number of particles used for SVGD typically varies between 10 and 100, in comparison to particle sets of size $> 10,000$ generated by MCMC chains. Therefore SVGD addresses the sample complexity challenge, but not necessarily the training time challenge.

2.3.3 Stochastic Variational Inference (SVI)

Stochastic Variational Inference accelerates the Bayesian learning process by reformulating the Bayesian learning problem as a parameterized optimization problem. This reformulation accelerates training and leads to low-complexity representations. However the strong assumptions employed by SVI can degrade the quality of the posterior approximation q.

Let $\boldsymbol{\theta}$ be the parameters to infer and let $q(\boldsymbol{\theta})$ be the approximating distribution to the target posterior distribution

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}).$$

SVI [69] solves an optimization problem where the loss function is the KL divergence and the

goal is to find the best approximating density $q^\star$ among a parameterized class of densities $\mathcal{P}$:

$$q^\star(\boldsymbol{\theta}) = \underset{q(\boldsymbol{\theta})\in\mathcal{P}}{\arg\min}\, \mathrm{KL}\left(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})\right), \quad \mathrm{KL}\left(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})\right) = \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathcal{D})}\right]. \tag{2.16}$$

The KL divergence can be re-written as

$$\begin{aligned} \mathrm{KL}\left(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})\right) &= \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log q(\boldsymbol{\theta}) - \log p(\mathcal{D}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})\right] + \text{const.} \\ &= -\mathbb{E}_{q(\boldsymbol{\theta})}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] + \mathrm{KL}\left(q(\boldsymbol{\theta})||p(\boldsymbol{\theta})\right) + \text{const.} \end{aligned} \tag{2.17}$$

This is a combination of the log-likelihood (model fit) and the divergence from the approximate posterior to the prior (low-rank). To approximate the expectation in the log-likelihood term one samples from the variational distribution q. The KL-divergence is either approximated via sampling or evaluated in a closed form. The form in Equation (2.17) requires the evaluation of the full-data model likelihood. If the data is large the full-data likelihood $p(\mathcal{D}|\boldsymbol{\theta})$ is intractable, so we approximate the likelihood by subsampling a minibatch $\mathcal{M} \subset \mathcal{D}$.

The objective in Equation 2.17 admits fast gradient-based optimization which speeds up training compared to nonparametric particle-based methods. However the parametric form of q can reduce the quality of the posterior representation.

Chapter 3

Stein Variational Gradient Descent for Compact End-to-End Training in TT Format

Tensor decomposition is an effective approach to compress over-parameterized neural networks and to enable their deployment on resource-constrained hardware platforms. However, directly applying tensor compression in the training process is a challenging task due to the aforementioned difficulty of choosing a proper tensor rank. In order to address this challenge, this chapter proposes a low-rank Bayesian tensorized neural network. Our Bayesian method performs automatic model compression via an adaptive tensor rank determination. We also present approaches for posterior density calculation and maximum a posteriori (MAP) estimation for the end-to-end training of our tensorized neural network. We provide experimental validation on a two-layer fully connected neural network, a 6-layer CNN and a 110-layer residual neural network where our work produces 7.4× to 137× more compact neural networks directly from the training while achieving high prediction accuracy.

Contributions. Inspired by the recent Bayesian CP and Tucker tensor completion [58,62], we develop a novel low-rank Bayesian tensorized neural network. Our contribution is two-fold. Firstly, we present a Bayesian model to compress the model parameters (e.g., weight matrices and convolution kernels) via tensor train decomposition [87]. We develop the first method for Bayesian tensor rank determination in nonlinear models such as neural networks. Our method employs a proper prior density to automatically determine the tensor ranks

based on the given training data, which is beyond the capability of existing tensorized neural networks. Secondly, we develop training algorithms to estimate the full posterior density and the MAP point. To solve the large-scale Bayesian inference problem we approximate the posterior density by Stein variational gradient descent [88]. The proposed framework can generate much more compact neural networks. Our method can also provide uncertainty estimation, which is important for certain applications like decision making in autonomous driving [26] and robotics. Code to reproduce our experiments is publicly available at `https://github.com/colehawkins/svgd-tensor-neurocomputing`

3.1 Proposed Bayesian Model

3.1.1 Low-Rank Bayesian Tensorized Neural Networks (LR-BTNN)

Let $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ be the training data, where $\mathbf{y}_i \in \mathbb{R}^S$ is an output label. We intend to train an L-layer tensorized neural network

$$\mathbf{y} \approx \mathbf{f}\left(\mathbf{x} \mid \mathbf{\Psi}\right) = \mathbf{g}\left(\mathbf{x}| \left\{\boldsymbol{\mathcal{W}}^{(l)}\right\}_{l=1}^L\right). \tag{3.1}$$

as described in (2.8). Here $\boldsymbol{\mathcal{W}}^{(l)}$ is an unknown tensor describing the massive model parameters in the l-th layer. We consider two kinds of tensorized layers in this paper:

- **TT-FC Layer.** In a fully connected (FC) layer, a vector is mapped to a componentwise nonlinear activation function by a weight matrix $\mathbf{W}^{(l)}$. We tensorize $\mathbf{W}^{(l)}$ as $\boldsymbol{\mathcal{W}}^{(l)}$ according to (2.5).

- **TT-Conv Layer.** A convolutional filter takes the form $\boldsymbol{\mathcal{K}}^{(l)} \in \mathbb{R}^{t\times t\times C\times S}$ where $t \times t$ is the filter size, C and S are the numbers of input and output channels, respectively. The number of channels C and S are often larger than the filter size t, so we factorize $C = \prod_{i=1}^d c_i, S = \prod_{i=1}^d s_i$, reshape $\boldsymbol{\mathcal{K}}^{(l)}$ into a $t^2 \times c_1 \times \cdots \times c_d \times s_1 \times \cdots \times s_d$ tensor, and denote the reshaped tensor as $\boldsymbol{\mathcal{W}}^{(l)}$. Reshaping achieves better compression ratios when

C or S is large [51].

Our goal is to parameterize and compress each unknown $\boldsymbol{\mathcal{W}}^{(l)}$ in TT-format in the training process. To simplify notations, we consider a single-layer neural network parametrized by a single tensor $\boldsymbol{\mathcal{W}}$, but our results can be easily generalized to a general L-layer network (see our result section). In order to build a Bayesian model, we assume the following likelihood function

$$p\left(\boldsymbol{\mathcal{D}}|\left\{\boldsymbol{\mathcal{G}}^{(k)}\right\}_{k=1}^{d}\right)=\prod_{i=1}^{N}p\left(\mathbf{y}_i,\mathbf{g}\left(\mathbf{x}_i\mid[\![\boldsymbol{\mathcal{G}}^{(1)},\ldots,\boldsymbol{\mathcal{G}}^{(d)}]\!]\right)\right). \tag{3.2}$$

Here $\boldsymbol{\mathcal{G}}^{(k)}\in\mathbb{R}^{R_{k-1}\times M_k\times J_k\times R_k}$ is an unknown TT core of $\boldsymbol{\mathcal{W}}$, and the maximum TT-rank $\mathbf{R}=(1,R_1,\ldots,R_{d-1},1)$ sets an **upper bound** for the TT rank. The actual TT rank will be determined later. In this paper we focus on classification problems so we assume a multinomial likelihood. Let $y_{i,s}$ and g_s be the s-th component of $\mathbf{y}_i$ and $\mathbf{g}$, respectively, then we have

$$p\left(\mathbf{y}_i,\mathbf{g}\left(\mathbf{x}_i\mid[\![\boldsymbol{\mathcal{G}}^{(1)},\ldots,\boldsymbol{\mathcal{G}}^{(d)}]\!]\right)\right)=\prod_{s=1}^{S}g_s\left(\mathbf{x}_i\mid[\![\boldsymbol{\mathcal{G}}^{(1)},\ldots,\boldsymbol{\mathcal{G}}^{(d)}]\!]\right)^{y_{i,s}} \tag{3.3}$$

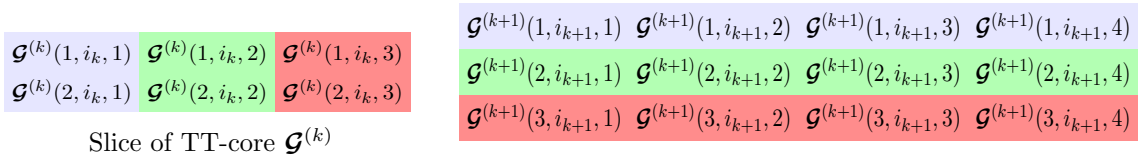
As a result, the negative log-likelihood is the standard cross-entropy loss

$$\mathcal{L}(\{\boldsymbol{\mathcal{G}}^{(k)}\})=-\sum_{i=1}^{N}\sum_{s=1}^{S}y_{i,s}\log g_s\left(\mathbf{x}_i\mid[\![\boldsymbol{\mathcal{G}}^{(1)},\ldots,\boldsymbol{\mathcal{G}}^{(d)}]\!]\right). \tag{3.4}$$

In order to infer the unknown TT cores $\{\boldsymbol{\mathcal{G}}^{(k)}\}_{k=1}^{d}$ and to decide the actual ranks, we will further introduce some hidden variables $\{\boldsymbol{\lambda}^{(k)}\}_{k=1}^{d-1}$ to parameterize the prior density of $\{\boldsymbol{\mathcal{G}}^{(k)}\}_{k=1}^{d}$ (which will be explained in Section 3.1.2). Let $\boldsymbol{\theta}$ denote all unknown variables

$$\boldsymbol{\theta}:=\left\{\left\{\boldsymbol{\mathcal{G}}^{(k)}\right\}_{k=1}^{d},\{\boldsymbol{\lambda}^{(k)}\}_{k=1}^{d-1}\right\} \tag{3.5}$$

which is described with a prior density $p(\boldsymbol{\theta})$. Then we can build a tensorized neural network by



Slice of TT-core $\mathcal{G}^{(k)}$ Slice of TT-core $\mathcal{G}^{(k+1)}$

Figure 3.1: Elements of 3-way $\mathcal{G}^{(k)}$ and $\mathcal{G}^{(k+1)}$. The elements controlled by the same entry of $\boldsymbol{\lambda}^{(k)}$ are marked with the same color. Here the upper bound of TT rank is set as $R_{k-1} = 2, R_k = 3, R_{k+1} = 4$.

estimating the MAP point or the full distribution of the following posterior density function:

$$p(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}}) = \frac{p(\boldsymbol{\mathcal{D}}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{\mathcal{D}})} \propto p(\boldsymbol{\mathcal{D}}|\boldsymbol{\theta})p(\boldsymbol{\theta}) = p(\boldsymbol{\mathcal{D}}, \boldsymbol{\theta}). \tag{3.6}$$

There exist two key challenges. Firstly, how shall we choose the prior density $p(\boldsymbol{\theta})$ to ensure desired model structures? Secondly, how can we efficiently solve the resulting large-scale Bayesian inference?

3.1.2 Selection of Prior Density Functions

In order to achieve automatic model compression in the training, the prior density function in (3.6) should be chosen such that: (1) the actual rank of $\mathcal{G}^{(k)}$ is very small; (2) manual rank tuning can be avoided.

Prior Density for $\mathcal{G}^{(k)}$. We specify the prior density on a TT-matrix core. The size of $\mathcal{G}^{(k)}$ is fixed as $R_{k-1} \times M_k \times J_k \times R_k$. In order to reduce the TT rank, we will enforce some rows and columns in the slice $\mathcal{G}^{(k)}(:, m_k, j_k, :)$ to zero. The main challenge is that the matrix products are coupled: the r_k-th column of $\mathcal{G}^{(k-1)}(:, m_{k-1}, j_{k-1}, :)$ and the r_k-th row of $\mathcal{G}^{(k)}(:, m_k, j_k, :)$ should simultaneously shrink to zero if a rank deficiency happens. Fig. 3.1 uses the slices of two adjacent 3-way TT cores to show this coupling in the TT decomposition.

In order to address the above challenge, we extend the sparsity-enforcing priors from [58] which were developed for CP and Tucker tensor completion but are not applicable to TT format. Specifically, we introduce the non-negative vector parameter $\boldsymbol{\lambda}^{(k)} = [\lambda_1^{(k)}, \ldots, \lambda_{R_k}^{(k)}] \in \mathbb{R}^{R_k}$ to

control the actual rank $\hat{R}_k$ for each k with $1 \leq k \leq d-1$. For each intermediate TT core $\boldsymbol{\mathcal{G}}^{(k)}$, we place a normal prior on its entries:

$$p\left(\boldsymbol{\mathcal{G}}^{(k)} \mid \boldsymbol{\lambda}^{(k-1)}, \boldsymbol{\lambda}^{(k)}\right) = \prod_{r_{k-1}, m_k, j_k, r_k} \mathcal{N}\left(\boldsymbol{\mathcal{G}}^{(k)}(r_{k-1}, m_k, j_k, r_k) \mid 0, \lambda_{r_{k-1}}^{(k-1)} \cdot \lambda_{r_k}^{(k)}\right) \tag{3.7}$$

where $2 \leq k \leq d-1$, $1 \leq r_{k-1} \leq R_{k-1}, 1 \leq r_k \leq R_k, 1 \leq m_k \leq M_k$ and $1 \leq j_k \leq J_k$.

Because $R_0 = R_d = 1$, the ranks of the first and last TT cores are controlled by only one vector. We place a similar normal prior on each:

$$\begin{aligned} p\left(\boldsymbol{\mathcal{G}}^{(1)} \mid \boldsymbol{\lambda}^{(1)}\right) &= \prod_{m_1, j_1, r_1} \mathcal{N}\left(\boldsymbol{\mathcal{G}}^{(1)}(1, m_1, j_1, r_1) \mid 0, \left(\lambda_{r_1}^{(1)}\right)^2\right) \\ p\left(\boldsymbol{\mathcal{G}}^{(d)} \mid \boldsymbol{\lambda}^{(d-1)}\right) &= \prod_{r_{d-1}, m_d, j_d} \mathcal{N}\left(\boldsymbol{\mathcal{G}}^{(d)}(r_{d-1}, m_d, j_d, 1) \mid 0, \left(\lambda_{r_{d-1}}^{(d-1)}\right)^2\right). \end{aligned} \tag{3.8}$$

Here we use the squares $\left(\lambda_{r_1}^{(1)}\right)^2$ and $\left(\lambda_{r_{d-1}}^{(d-1)}\right)^2$ to ensure that the order of magnitude of the priors is consistent across all TT cores. To apply the same prior to the standard tensor train we remove the third index j_k of each TT core.

Prior Density for $\boldsymbol{\lambda}^{(k)}$. In order to avoid tuning $\{\boldsymbol{\lambda}^{(k)}\}$ and TT-ranks manually, we set each $\boldsymbol{\lambda}^{(k)}$ as a random vector and impose a gamma prior on its entries:

$$p(\boldsymbol{\lambda}^{(k)}) = \prod_{r_k=1}^{R_k} \mathrm{Ga}(\lambda_{r_k}^{(k)} | a_\lambda, b_\lambda). \tag{3.9}$$

The hyperparameters are set to $a_\lambda = 1, b_\lambda = 5$ to encourage sparsity. We chose the Gamma prior over other sparsity-inducing priors (i.e. Normal, Laplace, Horseshoe) due to better empirical compression ratios. We also experimented with shrinkage priors that were not coupled across adjacent tensor cores and found that they gave poor empirical compression.

Overall Prior for $\boldsymbol{\theta}$. Combining (3.7), (3.8) and (3.9), we have the overall prior density

$p(\boldsymbol{\theta})$:

$$p\left(\boldsymbol{\theta}\right) = p\left(\boldsymbol{\mathcal{G}}^{(1)} \mid \boldsymbol{\lambda}^{(1)}\right) p\left(\boldsymbol{\mathcal{G}}^{(d)} \mid \boldsymbol{\lambda}^{(d-1)}\right) \prod_{k=2}^{d-1} p\left(\boldsymbol{\mathcal{G}}^{(k)} \mid \boldsymbol{\lambda}^{(k-1)}, \boldsymbol{\lambda}^{(k)}\right) \prod_{k=1}^{d-1} p(\boldsymbol{\lambda}^{(k)}). \tag{3.10}$$

Rank-Shrinkage Effect. We illustrate the rank-shrinkage of $\boldsymbol{\lambda}$ in our low-rank tensor prior with a close examination of the conditional prior density of the slices of the first core tensor $p(\boldsymbol{\mathcal{G}}^{(1)}(:,:,:,r_k)|\lambda_{r_k}^{(1)})$. Other tensor cores are similar. We observe that

$$\begin{aligned} p\left(\boldsymbol{\mathcal{G}}^{(1)}(:,:,:,r_k) \mid \lambda_{r_k}^{(1)}\right) &= \prod_{m_1,j_1} \mathcal{N}\left(\boldsymbol{\mathcal{G}}^{(1)}(1,m_1,j_1,r_k) \mid 0, \left(\lambda_{r_k}^{(1)}\right)^2\right) \\ &= \prod_{m_1,j_1} \frac{1}{\lambda_{r_k}^{(1)}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\boldsymbol{\mathcal{G}}^{(1)}(1,m_1,j_1,r_k)}{\lambda_{r_k}^{(1)}}\right)^2}. \\ &= \left(\frac{1}{\lambda_{r_k}^{(1)}\sqrt{2\pi}}\right)^{M_1 J_1} e^{-\frac{1}{2}\left(\frac{\|\boldsymbol{\mathcal{G}}^{(1)}(:,:,:,r_k)\|_2}{\lambda_{r_k}^{(1)}}\right)^2}. \end{aligned} \tag{3.11}$$

We will apply standard results for the Mahalanobis distance

$$d^2 = \left(\frac{\|\boldsymbol{\mathcal{G}}^{(1)}(:,:,:,r_k)\|_2}{\lambda_{r_k}^{(1)}}\right)^2 \tag{3.12}$$

to demonstrate how shrinkage in λ_{r_k} leads to shrinkage in $\boldsymbol{\mathcal{G}}^{(1)}(:,:,:,r_k)$. The Mahalanobis d distance follows a χ^2 distribution so

$$p(d^2 < \epsilon) = \frac{1}{2^{M_1 J_1}\Gamma(M_1 J_1/2)} \epsilon^{M_1 J_1/2-1} e^{-\frac{\epsilon}{2}}. \tag{3.13}$$

Finally, we manipulate Equation (3.13) by plugging in the value of d^2 from Equation (3.12) to get

$$p\left(\|\boldsymbol{\mathcal{G}}^{(1)}(:,:,:,r_k)\|_2 < \sqrt{\epsilon}\lambda_{r_k}^{(1)}\right) = \frac{1}{2^{M_1 J_1}\Gamma(M_1 J_1/2)} \epsilon^{M_1 J_1/2-1} e^{-\frac{\epsilon}{2}}. \tag{3.14}$$

The right-hand side of Equation (3.14) is constant and independent of $\lambda_{r_k}^{(1)}$. Therefore, as $\lambda_{r_k}^{(1)} \to 0$, with high probability the tensor slice norm $\|\boldsymbol{\mathcal{G}}^{(1)}(:,:,:,r_k)\|_2 \to 0$.

3.1.3 Complete Probabilistic Model

Now we are ready to obtain the full posterior density by combining (3.2),(3.6) and (3.10):

$$
\begin{aligned}
p(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}}) =& \frac{1}{p(\boldsymbol{\mathcal{D}})} \prod_{i=1}^{N} p\left(\mathbf{y}_i, \mathbf{g}\left(\mathbf{x}_i \mid [\![\boldsymbol{\mathcal{G}}^{(1)}, \ldots, \boldsymbol{\mathcal{G}}^{(d)}]\!]\right)\right) p\left(\boldsymbol{\mathcal{G}}^{(1)} \mid \boldsymbol{\lambda}^{(1)}\right) \times \\
& p\left(\boldsymbol{\mathcal{G}}^{(d)} \mid \boldsymbol{\lambda}^{(d-1)}\right) \prod_{k=2}^{d-1} p\left(\boldsymbol{\mathcal{G}}^{(k)} \mid \boldsymbol{\lambda}^{(k-1)}, \boldsymbol{\lambda}^{(k)}\right) \prod_{k=1}^{d-1} p(\boldsymbol{\lambda}^{(k)}).
\end{aligned}
\tag{3.15}
$$

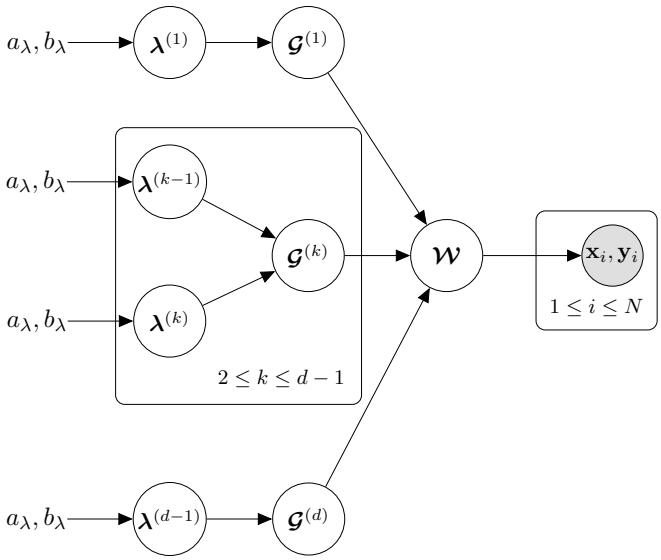


Figure 3.2: Bayesian graphical model for a low-rank Bayesian tensorized neural network parametrized by a single low-rank tensor $\boldsymbol{\mathcal{W}}$. There are N training samples.

The associated probabilistic graphical model is shown in Fig. 3.2. The user-defined parameters $a_{\boldsymbol{\lambda}}$ and $b_{\boldsymbol{\lambda}}$ generate a Gamma distribution for $\boldsymbol{\lambda}^{(k)}$ which tunes the actual rank of each TT core $\boldsymbol{\mathcal{G}}^{(k)}$ via a Gaussian distribution. The total number of parameters to be inferred for a single-layer tensorized neural network is $\sum_{k=1}^{d} M_k J_k R_{k-1} R_k + \sum_{k=1}^{d-1} R_k$. The extension to the an L-layer neural network is straightforward: one just needs to replicate the whole diagram except the training data by L times.

3.1.4 Automatic Rank Determination

The actual TT rank is determined by both the prior and training data. As shown in (3.7) and (3.8), each entry of $\boldsymbol{\lambda}^{(k)}$ directly controls one sub-tensor of $\boldsymbol{\mathcal{G}}^{(k)}$ and one sub-tensor of $\boldsymbol{\mathcal{G}}^{(k+1)}$. If the entry $\boldsymbol{\lambda}_{r_k}^{(k)}$ is large then elements of subtensors $\boldsymbol{\mathcal{G}}^{(k)}(:,:,:,r_k)$ and $\boldsymbol{\mathcal{G}}^{(k+1)}(r_k,:,:,:)$ can vary freely based on the training data. In contrast, if $\boldsymbol{\lambda}_{r_k}^{(k)}$ is close to zero, then the elements of $\boldsymbol{\mathcal{G}}^{(k)}(:,:,:,r_k)$ and $\boldsymbol{\mathcal{G}}^{(k+1)}(r_k,:,:,:)$ are more likely to vanish. Let $\bar{\boldsymbol{\lambda}}^{(k)}$ be the posterior mean of $\boldsymbol{\lambda}^{(k)}$ decided by both the prior and training data. Then the inferred TT-rank $\hat{\mathbf{R}} = [1, \hat{R}_1, \hat{R}_2, \ldots, \hat{R}_{d-1}, 1]$ is estimated as the number of nonzero elements in $\bar{\boldsymbol{\lambda}}^{(k)}$:

$$\hat{R}_k = \text{nnz}\left(\bar{\boldsymbol{\lambda}}^{(k)}\right) \text{ for } k = 1, 2, \ldots, d-1. \tag{3.16}$$

In practice, an element of $\bar{\boldsymbol{\lambda}}^{(k)}$ is regarded as zero if it is below the threshold $1e-2$. Such an automatic rank tuning reduces the actual number of model parameters in a single layer to $\sum_{k=1}^{d} M_k J_k \hat{R}_{k-1} \hat{R}_k$.

3.2 Bayesian and MAP Training

Now we describe how to train our low-rank Bayesian tensorized neural networks. We note here that prior work on Bayesian tensor rank determination [57, 58, 60, 62] is only suitable for linear models, and cannot generalize to highly nonlinear Bayesian tensorized neural networks. We demonstrate how to overcome this difficulty.

3.2.1 Full Bayesian Training

Existing Bayesian low-rank tensor methods either rely on mean-field variational inference or MCMC sampling. Mean-field approximations require that the tensor model is linear, and MCMC sampling is prohibitively expensive for large-scale or deep neural networks. Therefore, we employ the Stein variational gradient descent (SVGD) recently developed by [88] to provide a nonparametric approximation the posterior density $p(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}})$. The update from (2.15) requires the

gradient of the log-posterior with respect to the model parameters. The gradient $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}})$ is expressed as

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}}) = \sum_{i=1}^{N} \sum_{s=1}^{S} y_{i,s} \frac{\nabla_{\boldsymbol{\theta}} \left[g_s \left(\mathbf{x}_i \mid [\![\boldsymbol{\mathcal{G}}^{(1)}, \ldots, \boldsymbol{\mathcal{G}}^{(d)}]\!] \right) \right]}{g_s \left(\mathbf{x}_i \mid [\![\boldsymbol{\mathcal{G}}^{(1)}, \ldots, \boldsymbol{\mathcal{G}}^{(d)}]\!] \right)} + \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) \tag{3.17}$$

for classification. The first term is exactly the gradient of a maximum-likelihood tensorized training, and it is replaced by a stochastic gradient if N is large. The 2nd term caused by our low-rank prior is our only overhead over standard tensorized training [50], and does not require forming the full tensor. Let $\boldsymbol{\Lambda}_k = \operatorname{diag}\left(\boldsymbol{\lambda}^{(k-1)} \otimes \boldsymbol{\lambda}^{(k)}\right)$ be a diagonal matrix, and $\operatorname{vec}\left(\boldsymbol{\mathcal{G}}^{(k)}(:, m_k, j_k, :)\right)$ be the column-major vectorization. The gradients of the log-prior are provided below:

$$\begin{aligned}
\frac{\partial \log p(\boldsymbol{\theta})}{\partial \operatorname{vec}\left(\boldsymbol{\mathcal{G}}^{(k)}(:, m_k, j_k, :)\right)} &= -\boldsymbol{\Lambda}_k^{-1} \operatorname{vec}\left(\boldsymbol{\mathcal{G}}^{(k)}(:, m_k, j_k, :)\right) \\
\frac{\partial \log p(\boldsymbol{\theta})}{\partial \lambda_l^{(k)}} &= -\frac{1}{2} \sum_{m_{k+1}, j_{k+1}, r_{k+1}} \left\{ \frac{1}{\lambda_l^{(k)} \lambda_{r_{k+1}}^{(k+1)}} - \left(\frac{\boldsymbol{\mathcal{G}}^{(k+1)}(l, m_{k+1}, j_{k+1}, r_{k+1})}{\lambda_l^{(k)} \lambda_{r_{k+1}}^{(k+1)}} \right)^2 \right\} \\
&\quad - \frac{1}{2} \sum_{r_{k-1}, m_k, j_k} \left\{ \frac{1}{\lambda_{r_{k-1}}^{(k-1)} \lambda_l^{(k)}} - \left(\frac{\boldsymbol{\mathcal{G}}^{(k)}(r_{k-1}, m_k, j_k, l)}{\lambda_{r_{k-1}}^{(k-1)} \lambda_l^{(k)}} \right)^2 \right\} \\
&\quad + \frac{(a_\lambda - 1)}{\lambda_l^{(k)}} - b_\lambda.
\end{aligned} \tag{3.18}$$

3.2.2 MAP Training

To obtain the MAP estimation we run stochastic gradient descent to minimize the negative log-posterior:

$$\begin{aligned}
-\log p(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}}) = &-\sum_{i=1}^{N} \sum_{s=1}^{S} y_{i,s} \log g_s \left(\mathbf{x}_i \mid [\![\boldsymbol{\mathcal{G}}^{(1)}, \ldots, \boldsymbol{\mathcal{G}}^{(d)}]\!] \right) \\
&- \log p\left(\boldsymbol{\theta}\right) + \log p(\mathcal{D})
\end{aligned} \tag{3.19}$$

The local maximum achieved by MAP training provides a single non-Bayesian tensorized neural network that has been compressed by rank determination. This method is useful in order to quickly produce a single compressed model, but does not enable uncertainty quantification.

3.2.3 Initialization

Training deep neural networks requires careful initialization [9]. The same is true for tensorized neural networks [96]. A good empirically determined initialization distribution for the full tensor weights is $\mathcal{N}\left(0, \sqrt{\frac{2}{Q}}\right)$ where Q is the total number of parameters in the full tensor. In order to achieve variance $\sqrt{\frac{2}{Q}}$ for a low-rank TT or TT-matrix with ranks $R_1 = \cdots = R_{d-1} = R$, we initialize the TT core elements using a $\mathcal{N}\left(0, \sigma^2\right)$ with $\sigma^2 = \left(\frac{2}{Q}\right)^{1/2d} R^{1/d-1}$. This initialization provides a correction to the tensor core initialization described in [96] which contains an error.

3.3 Numerical Experiments

3.3.1 Experimental Setup

We validate our method using three network structures and three datasets. We use the Adam optimization algorithm [97] to initialize the first particle at the MAP point by minimizing Equation (3.19) and then run 5000 iterations of SVGD. For all trainable weights except the low-rank tensors in our proposed model we apply a $\mathcal{N}(0, 100)$ prior which acts as a weak regularizer. We refer to our proposed low-rank Bayesian tensorized model as "**LR-BTNN**", a Bayesian tensorized neural network with a $\mathcal{N}(0, 100)$ prior on all weights and convolution kernels as "**BTNN**", and a Bayesian non-tensorized neural network with a $\mathcal{N}(0, 100)$ prior on all parameters as "**BNN**". We use the threshold 10^{-2} on all $\lambda_j^{(k)}$ to truncate a TT-rank. In all experiments the maximum rank R of the proposed LR-BTNN model is the same as the rank of the fixed-rank BTNN model.

- **Toy Model (2 FC Layers).** First we test on the MNIST and Fashion-MNIST datasets [4, 98] using a network with two fully-connected layers. A similar tensorized neural network was studied in [52]. We compare the accuracy and rank determination ability of our approach as compared to the deterministic training approach from [52]. The first layer is size 784×625 with ReLU activation, and the second layer is size 625×10 with a soft-

max activation. Both layers contain a bias parameter. We convert the first layer into a TT-matrix with $(m_1, m_2, m_3, m_4) = (7, 4, 7, 4), (j_1, j_2, j_3, j_4) = (5, 5, 5, 5)$ and the second layer into a TT-matrix with $(m_1, m_2) = (25, 25)$ and $(j_1, j_2) = (5, 2)$. Both layers have maximum TT-rank $R_1 = \cdots = R_{d-1} = 20$. We initialize by training for 100 epochs on the log-posterior. We use 50 particles to approximate the posterior density.

- **Baseline Six Layer CNN (4 Conv layers + 2 FC layers).** We test on the CIFAR-10 dataset [99] using a baseline tensorized convolutional model from [50]. This CNN model consists of (Conv-128,BN,ReLU), (Conv-128,BN,ReLU), max-pool 3×3, (Conv-256,BN,ReLU), (Conv-256,BN,ReLU), max-pool 5×5, fc-512, fc-10. All convolutions are 3×3 with stride 1. Following [51] we do not tensorize the first convolutional layer, which contains less than 1% of the parameters. We set the maximum TT-ranks of all layers to 30. We extend the original training 100 epoch training schedule from [51] to 120 epochs to account for the more complex log-posterior loss function. We use 20 particles for the SVGD fully Bayesian estimation.

- **ResNet-110 (109 Conv layers + 1 FC layer).** We further test the baseline Keras ResNetv1 structure [1, 100] on the CIFAR-10 datasets. We do not tensorize the convolutions in the first ResBlock (first 36 layers) or the 1×1 convolutions. For all other convolutions we set the maximum TT-rank to 20. We use the standard 200 epoch training schedule to find the MAP point. As in the previous CNN experiment, we also use 20 particles for the SVGD fully Bayesian estimation.

3.3.2 Results

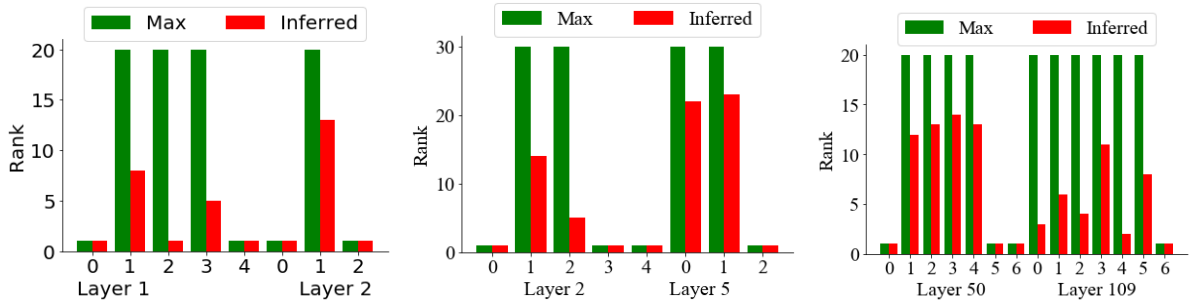
Table 3.1 shows the overall performances of our LR-BTNN with BNN and BTNN on the three examples. We evaluate test accuracy of the single-particle map estimate (MAP Acc.) and the uncertainty quality of the fully Bayesian SVGD model (test LL). From the table, we can compare the following performance metrics:

Table 3.1: Results of a standard Bayesian neural network (BNN), a Bayesian tensorized neural network (BTNN) with Gaussian prior, and our low-rank Bayesian tensorized neural network (LR-BTNN). We report test log-likelihood for the SVGD models and test accuracy (Acc.) for the MAP models.

Example	Model	SVGD Param #	MAP Param #	Log-Lik.	Acc.
Toy Model (2 FC, MNIST)	BNN	24,844,250	496,885	-0.193	97.6%
	BTNN	1,361,750	27,235	-0.188	97.7%
	LR-BTNN	**181,250 (137×)**	**3,625 (137×)**	**-0.106**	**97.8%**
Toy Model (2 FC, F-MNIST)	BNN	24,844,250	496,885	-0.619	87.1 %
	BTNN	1,361,750	27,235	-0.847	86.7%
	LR-BTNN	**642,750 (38.6×)**	**12,375 (40.1×)**	**-0.410**	**87.7%**
Baseline CNN (4 Conv+2 FC)	BNN	31,388,360	1,569,418	-0.497	**89.0%**
	BTNN	13,737,360	686,868	**-0.463**	88.8%
	LR-BTNN	**1,987,520 (15.8×)**	**99,376 (15.8×)**	-0.471	87.4%
ResNet-110 (109 Conv+1 FC)	BNN	34,855,240	1,742,762	**-0.506**	**92.6%**
	BTNN	12,895,800	644,790	-0.521	91.1%
	LR-BTNN	**4,733,028 (7.4×)**	**236,651 (7.4×)**	-0.515	90.4%

Model Size. The 3rd and 4th columns of Table 3.1 show the number of model parameters in the full posterior density estimations and in the MAP estimations, respectively. Because of the automatic tensor rank reduction, our LR-BTNN method generates much more compact neural networks. The compression ratio is very high when the networks have FC layers only (137× reduction over BNN on the MNIST example). The compression ratio becomes smaller when the network has more convolution layers (e.g., 27.3× on the CNN and 7.4× on ResNet-110). Our method outperforms [48, 51] which compressed the convolution layers typically by 2× to 4×.

Prediction Accuracy. The 5th and 6th columns of Table 3.1 show the prediction accuracy of our fully Bayesian and MAP estimations, respectively. For the MAP estimation, the accuracy loss of our LR-BTNN model is very small, and our proposed model even achieves the best performance on the MNIST and Fashion-MNIST examples. For the fully Bayesian model, we measure the probabilistic model accuracy by computing the predictive log likelihood on held-out test data (denoted as "Test LL" in the table). Our LR-BTNN performs much better than other



(a) Two FC layers in the MNIST example.

(b) Conv (left) and FC (right) layers in CNN.

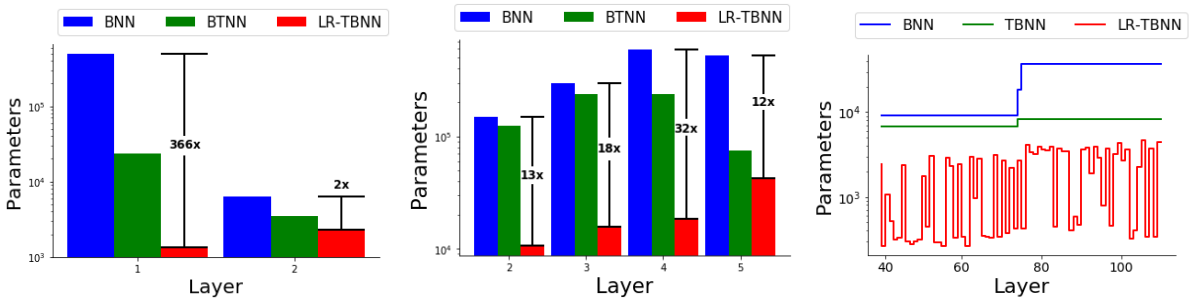
(c) Two Conv layers in ResNet-110.

Figure 3.3: Inferred TT-ranks at specified layers in the three examples.

two methods on the MNIST and Fashion-MNIST examples, and all models are comparable on the CIFAR-10 tasks despite the fact that our model is much smaller.

Automatic Rank Determination. The main advantage of our LR-BTNN method is its automatic rank determination and model compression in the traning process. Fig. 3.3 shows the rank determinations of some specific layers in all three examples. Fig. 3.4 shows the resulting layer-wise parameter reduction due to the automatic rank determination. The MNIST classification model has two overly parameterized FC layers, so the actual TT ranks are much lower than the maximum ranks. In fact, the TT-rank in Layer 1 reduces from $(1, 20, 20, 20, 1)$ to $(1, 8, 1, 5, 1)$, and the TT-rank in Layer 2 reduces from $(1, 20, 1)$ to $(1, 13, 1)$. This provides $8.5\times$ compression over the naive TNN and overall $138\times$ compression over the non-tensorized neural networks. A naive tensorization of the CNN model gives a compression ratio of $2.3\times$, and the rank determination gives a further parameter reduction of $6.9\times$, leading to an overall $15.8\times$ compression. Most layers in ResNet-110 are convolutions blocks with small numbers of filters, and there is only one small dense matrix. These facts make tensor compression less effective on this ResNet-110 example.

Uncertainty Quantification. Our LR-BTNN method is able to quantify the model uncertainty, which is critical for decision making in uncertain environments. Specifically, based on the posterior density obtained from SVGD, we can easily estimate the probability density of an predicted output. As an example, Fig. 3.5 shows an image that is hard to classify. The



(a) MNIST. (b) CNN. (c) ResNet-110.

Figure 3.4: Compression of parameters at different layers due to the automatic rank determination.

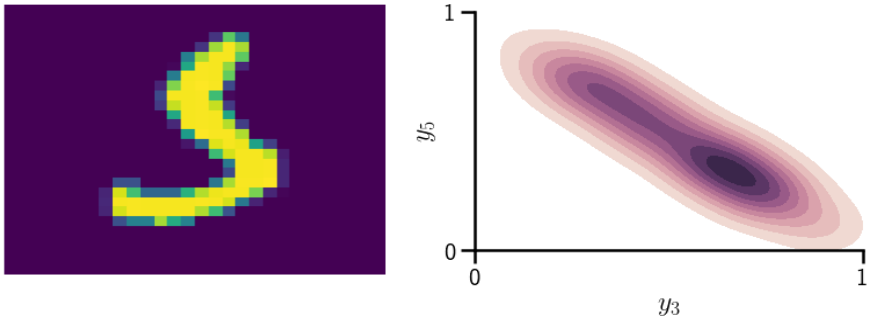


Figure 3.5: Left: a challenging input image with true label 3. Right: the joint marginal density of softmax output 3 (x-axis) and softmax output 5 (y-axis).

predicted density shows that this image can be recognized as "3" with the highest probability, but it can also be recognized as "5" with a high probability. A possible pitfall of SVGD is that all particles collapse to the MAP point [101]. This does not occur for our model, as the MAP log-likelihood value consistently outperforms the SVGD log-likelihood value.

3.3.3 Cross-Validation for Rank Selection

Cross-validation is a standard strategy for parameter tuning. However this approach is extremely time-consuming for selecting the tensor rank parameter because the rank parameter is discrete. Therefore combinatorial searches are required to select the best tensor ranks in the standard tensorized neural network training. In contrast, our model can determine the tensor rank in a single training run. We illustrate the effect of tuning the tensor rank in Table 3.2.

Table 3.2: Rank parameter tuning experiment on the MNIST dataset. BTNN-r is a tensorized neural network with the same MNIST architecture as before and fixed tensor rank r.

Model	Param #	Accuracy
BTNN-5	3,160	96.89
BTNN-10	8,435	97.23
BTNN-15	16,460	97.44
BTNN-20	27,235	97.68
BTNN-25	40,760	97.66
BTNN-30	57,035	97.63
LR-BTNN	**3,625**	**97.78**

We train the standard tensorized neural network (BTNN) on the MNIST task and vary the fixed tensor rank r of both layers to observe the parameter sensitivity. We observe that even after fine-tuning the tensor rank our model presents the most attractive parameter/accuracy tradeoff. This is because our model can automatically select non-uniform tensor ranks (i.e. (1,7,1,5,1)) without prohibitively expensive combinatorial search.

3.3.4 Particle Number Selection

For the MNIST model we found that existing SVGD approaches to non-tensorized training use $20 - 100$ particles for posterior approximation [88, 102] so we selected an intermediate value of 50. In Figure 3.6 we plot the particle/test log-likelihood tradeoff for each model (BNN,BTNN,LR-BTNN) on Fashion-MNIST. We observe that after approximately 40 particles the approximation quality is stable. For the larger CIFAR-10 experiments we selected the number of particles as 20 due to GPU memory constraints. We address the challenge of particle selection with another work in this thesis (see Chaper 5).

3.3.5 Rank Determination Overhead

To measure the overhead of our rank determination approach compared with the fixed-rank tensorized neural network training we measure the per-epoch timing overhead of the rank determination. The results are presented in Table 3.3. We observe that our proposed method

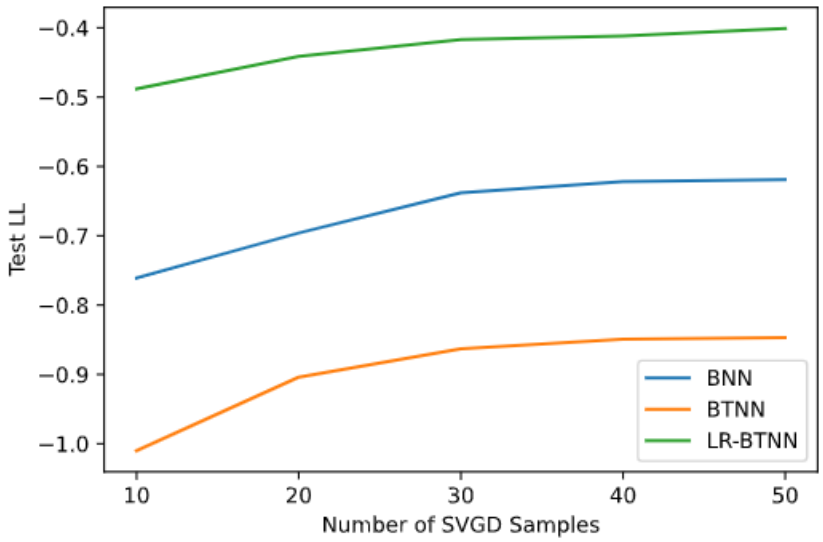


Figure 3.6: Test log-likelihood sensitivity vs. number of particles used for the Fashion-MNIST task.

(LR-BTNN) adds negligible overhead to standard fixed-rank training (BTNN).

Table 3.3: Rank determination training overhead. All results are reported in seconds per epoch for MAP training. BTNN does not use rank determination. Our proposed LR-BTNN method uses rank determination.

Task	BTNN	LR-BTNN
Toy Model (2 FC, MNIST)	9.7	9.8
Baseline CNN (4 Conv+2 FC)	56.1	56.3
Resnet-110 (109 Conv+1 FC)	207.2	207.5

3.4 Conclusion

We have proposed a low-rank Bayesian tensorized neural networks in the tensor train format. Our formulation provides an automatic rank determination and model compression in the end-to-end training. A Stein variational inference method has been employed to perform full Bayesian estimations, and the resulting model can predict output uncertainties. Our methods have shown excellent accuracy and model compression ratios on various neural network structures with both fully connected and convolution layers for the MNIST, Fashion-MNIST,

and CIFAR-10 data sets. Our method is shown to be more effective on FC layers than on convolution layers.

Chapter 4

Stochastic Variational Inference For Scalable End-to-End Compact Training in CP, TT, TTM, and Tucker Format

In comparison to the methods proposed in Chapter 3, the methods introduced in this chapter make the following improvements:

- Dramatically improving scalability by introducing a new Stochastic Variational Inference Bayesian solver, allowing us to demonstrate our method on much larger networks.
- Extending Bayesian techniques for rank determination in nonlinear tensor problems to all commonly used low-rank tensor formatst (CP, Tucker, TT, and TTM).

This chapter presents a novel end-to-end framework for low-rank tensorized training. We first develop a Bayesian model for automatic rank determination that supports various low-rank tensor formats (e.g., CP, Tucker, tensor train and Tensor-Train Matrix). This improves the method in Chapter 3 by including a wider range of low-rank tensor formats. Then we develop a customized Bayesian solver to train large-scale tensorized neural networks. This improves upon the scalability of the SVGD training presented in Chapter 3. Our training methods shows orders-of-magnitude parameter reduction and little accuracy loss (or even better accuracy) in the experiments. On a very large deep learning recommendation system with over 4.2×10^9 model parameters, our method can reduce the parameter number to 1.6×10^5 automatically in the training process (i.e., by 2.6×10^4 times) while achieving almost the same accuracy.

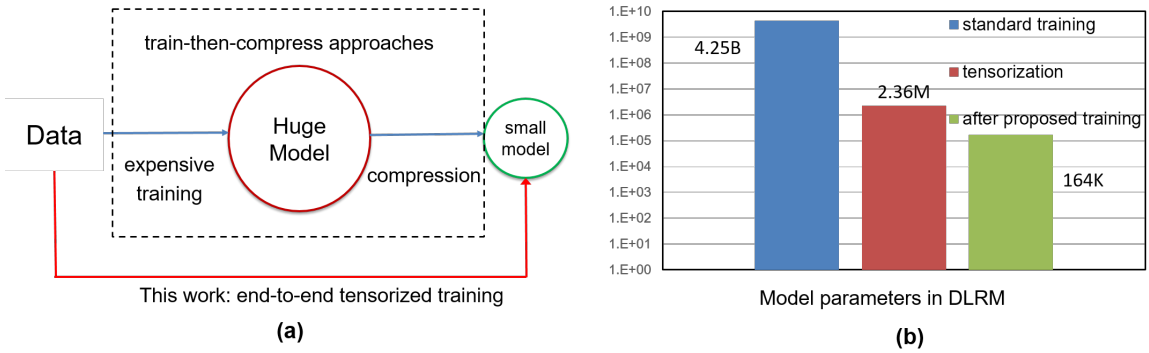


Figure 4.1: (a) Key idea of this work. Conventional train-then-compress approaches have high training costs. In constrast, the proposed end-to-end tensorized training can reduce the training variables significantly and directly produce ultra-compact neural networks. (b) Effectiveness of this approach on a realistic DLRM benchmark. Standard methods train 4.25 billion variables. Our proposed method only trains 2.36 million variables, which are further reduced to 164K in the training process due to the automatic tensor rank determination.

Contributions This paper will present a rank-adaptive end-to-end tensorized training method to generate ultra-compact neural networks directly from scratch. As shown in Fig. 4.1 (a), our method avoids the expensive full-size training in contrast with existing post-training tensor compression methods [47, 48, 51, 80]. Our method can reduce the training and inference variables by several orders of magnitude, and may achieve further reductions if combined with low-precision numerical operations [38,84,85,86]. This work can make a great practical impact: it may enable energy-efficient training of medium- or large-size neural networks on edge devices (e.g, embedded GPUs and FPGA), which is impossible to achieve at this moment with existing training methods. Some recent works have studied low-rank tensorized training [50, 52, 103], but they fix the tensor ranks before training. It is hard to decide a proper tensor rank parameter *a-priori* in practice, therefore one often has to perform extensive combinatorial searches and many training runs until a good rank parameter is found.

We make the following contributions to achieve efficient one-shot tensorized training:

- **A general-purpose rank-adaptive Bayesian tensorized model.** The training cost and model performance are controlled by tensor ranks, which are unknown *a priori*. In order to avoid expensive manual search for tensor ranks required by recent works [50, 52, 103], we develop a novel Bayesian model to determine both tensor ranks and factors

automatically. Existing tensor-based modeling methods are problem-specific and focus on a single tensor format [58, 62, 104, 105]. In contrast our work includes all four low-rank tensor formats in common use (CP, Tucker, tensor-train, and tensor-train matrix) and make general advances in low-rank tensor-based modeling. This paper focuses on neural networks, but our method can easily be applied to other tensor problems (e.g., tensor completion, tensor regression and multi-task tensor learning).

- **A scalable Stochastic Variational Inference Bayesian solver for the proposed tensorized neural networks.** Training Bayesian tensorized neural networks is expensive, and existing approaches incur high memory and compute requirements. This is because particle-based Bayesian methods require multiple model copies and multiple forward propagations for every training and inference step [105]. Existing mean-field Bayesian tensor completion solvers [58, 62, 104] do not work for tensorized neural networks because of the highly nonlinear forward propagation model in our case. In this work we improve the approximate Bayesian inference method [69]. Specifically, we observe that directly employing the solver in [69] causes large gradient variance in our tensorized model. Therefore, we simplify the posterior density of some rank-controlling hyper parameters, and develop an analytical/numerical hybrid approach for the solution update. This customized Bayesian solver infers the unknown tensor factors and tensor ranks of realistic neural networks in a single training run, enabling training and quantifying the uncertainty of extremely large-scale deep learning models that are beyond the capability of existing Bayesian solvers.

- **Extensive numerical validations.** We test our algorithms on four benchmarks with model parameters ranging from 4×10^5 to 4.2×10^9. Our method can reduce the training variables by several orders of magnitude with little or even no loss of accuracy. For instance, our method achieves $26,000\times$ parameter reduction when training a large-scale DLRM model as shown in Fig. 4.1 (b). We also compare our methods with existing tensorized neural network methods [50, 51, 52, 103] including post-training compression

and fixed-rank tensorized training, which clearly demonstrates the advantage of our rank-adaptive training method in terms of variable reduction and model accuracy.

To the best of our knowledge, this work is the first end-to-end Bayesian method that automatically determines the tensor rank in large-scale neural network training (with billions of model parameters) and supports multiple low-rank tensor formats simultaneously. This work will enable energy-efficient and low-cost training of realistic neural networks in resource-constrained scenarios such as internet of things (IoT), robotic systems and mobile phones, as demonstrated by our recent preliminary FPGA prototype for on-device training [18]. The Bayesian solution will enable uncertainty quantification of the prediction results, which is important in safety-critical applications such as autonomous driving and medical imaging. Code to reproduce our results is publicly available at `https://github.com/colehawkins/bayesian-tensor-rank-determination`.

4.1 Bayesian Low-Rank Tensorized Model

In this work, we plan to develop a tensorized training method that can automatically determine the tensor ranks in the training process. This method requires only one training run and avoids the high cost of uncompressed training. Bayesian methods have been employed for tensor completion and factorization [58, 62, 63], where the observed data is a linear function of tensor elements. However, existing Bayesian tensor solvers do not work for tensorized neural networks due to the nonlinear forward model and large number of unknown variables.

4.1.1 High-Level Bayesian Formulation

We first describe a general-purpose Bayesian model for training low-rank tensorized neural networks. For notational convenience we assume that our neural network $\mathbf{f}$ has one nonlinear layer, and that its weight matrix $\mathbf{W}$ is folded to a single tensor $\boldsymbol{\mathcal{A}}$. Extending our method to general multi-layer cases with multiple tensors is straightforward, and we will report results on general multi-layer models in Section 4.3.

Given a training data set $\mathcal{D}$, our goal is to determine the unknown low-rank factors $\mathbf{\Phi}$ for $\boldsymbol{\mathcal{A}}$, the associated tensor ranks, and the bias vector $\mathbf{b}$. We introduce hyper parameters $\mathbf{\Lambda}$ to control the tensor ranks and model complexity. Our posterior distribution is

$$p(\mathbf{\Psi}, \mathbf{\Lambda}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{\Psi})p(\mathbf{\Psi}, \mathbf{\Lambda})}{p(\mathcal{D})}, \text{ with } \mathbf{\Psi} = \{\mathbf{\Phi}, \mathbf{b}\}. \tag{4.1}$$

Here $p(\mathcal{D}|\mathbf{\Psi})$ is the model likelihood, $p(\mathbf{\Psi}, \mathbf{\Lambda})$ is the joint prior and $p(\mathcal{D})$ is the model evidence

$$p(\mathcal{D}) = \int_{\mathbf{\Psi}, \mathbf{\Lambda}} p(\mathcal{D}|\mathbf{\Psi})p(\mathbf{\Psi}, \mathbf{\Lambda}) d\mathbf{\Psi} d\mathbf{\Lambda}. \tag{4.2}$$

The likelihood and joint prior are specified below:

- **Likelihood function:** $p(\mathcal{D}|\mathbf{\Psi})$ and data $\mathcal{D}$ are determined by a forward propagation model. Let $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ be a training sample where $\mathbf{x}$ is the neural network input and $\mathbf{y}$ is the associated true label. The multinomial likelihood function for a neural network classifier with C potential classes is

$$p(\mathcal{D}|\mathbf{\Psi}) \propto \prod_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \prod_{c=1}^{C} \mathbf{f}(\mathbf{x}|\mathbf{\Psi})_c^{y_c}. \tag{4.3}$$

  where y_c is the correct class label. Here $\mathbf{f}$ is the forward propagation model in (2.9) which is conditioned on the given low-rank tensor factors and bias vectors. We omit the multinomial distribution constant of proportionality for simplicity.

- **Joint Prior:** We place an independent prior over the low-rank tensor factors and the bias term. We choose a weak normal prior for the bias term:

$$p(\mathbf{\Psi}, \mathbf{\Lambda}) = p(\mathbf{b})p(\mathbf{\Phi}, \mathbf{\Lambda}), \;\; p(\mathbf{b}) \propto \prod_i \frac{1}{\sigma_0^2} \exp\left(-\frac{b_i^2}{2\sigma_0^2}\right). \tag{4.4}$$

Here $p(\mathbf{\Phi}, \mathbf{\Lambda})$ is the joint prior for tensor factors $\mathbf{\Phi}$ and hyper parameters $\mathbf{\Lambda}$. The design of $p(\mathbf{\Phi}, \mathbf{\Lambda})$ depends on the tensor format we choose, which will be explained in Section 4.1.2 &

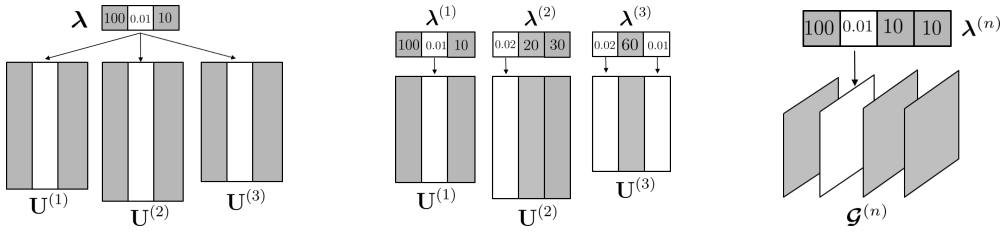


Figure 4.2: (a) For the CP prior, if one element of $\boldsymbol{\lambda}$ is small, one column is removed from every factor matrix. (b) For the Tucker prior, if one element of $\boldsymbol{\lambda}^{(n)}$ is small then one column of $\mathbf{U}^{(n)}$ shrinks to zero. (c) For the TT prior, if one element of $\boldsymbol{\lambda}^{(n)}$ is small then one slice of $\boldsymbol{\mathcal{G}}^{(n)}$ shrinks to zero. The columns/slices to be removed are marked in white.

4.1.3.

4.1.2 Tensor Factor Priors

Proper priors should be chosen in order to automatically shrink tensor ranks in the training process. Here we will specify the joint prior $p(\boldsymbol{\Phi}, \boldsymbol{\Lambda})$ for the four tensor formats described in Section 2.1: CP, Tucker, TT and TTM.

Firstly we specify the general form of $p(\boldsymbol{\Phi}, \boldsymbol{\Lambda})$. For the CP format, we initialize each factor $\mathbf{U}^{(n)}$ as a matrix with R columns. Assume that R is larger than the actual rank r, and all factors shrink to r columns in the training process. All CP factors have the same maximum rank (column number) so we use a single vector $\boldsymbol{\Lambda} = \boldsymbol{\lambda} \in \mathbb{R}^R$ to control the rank. The tensor rank in Tucker, TT or TTM format is a vector, and the rank associated with each mode can be different. Therefore, we require a collection of vectors $\boldsymbol{\Lambda} = \{\boldsymbol{\lambda}^{(n)}\}_{n=1}^d$ to control the ranks of each mode individually. Here $\boldsymbol{\lambda}^{(n)} \in \mathbb{R}^{R_n}$, and the "maximum rank" R_n exceeds the "actual rank" r_n of mode n. As a result, we introduce the general form

$$p(\boldsymbol{\Phi}, \boldsymbol{\Lambda}) = \begin{cases} p(\boldsymbol{\Phi}|\boldsymbol{\lambda})p(\boldsymbol{\lambda}) \text{ for CP format} \\ p(\boldsymbol{\Phi}|\{\boldsymbol{\lambda}^{(n)}\}) \prod_{n=1}^{d} p(\boldsymbol{\lambda}^{(n)}) \text{ for Tucker, TT \& TTM formats} \end{cases} \tag{4.5}$$

where the prior distribution(s) on $\boldsymbol{\lambda}$ or $\{\boldsymbol{\lambda}^{(n)}\}_{n=1}^d$ enforce(s) rank reduction.

Next we specify the tensor factor priors $p(\mathbf{\Phi}|\boldsymbol{\lambda})$ or $p(\mathbf{\Phi}|\{\boldsymbol{\lambda}^{(n)}\})$ for each tensor format, and we defer the prior on $\boldsymbol{\lambda}$ and $\{\boldsymbol{\lambda}^{(n)}\}_{n=1}^{d}$ to Section 4.1.3.

- **CP Format:** The CP tensor factors are d matrices $\mathbf{\Phi} = \{\mathbf{U}^{(n)}\}_{n=1}^{d}$. We assign a Gaussian prior with controllable variance to each element of each factor matrix $\mathbf{U}^{(n)}$:

$$p(\mathbf{\Phi}, \mathbf{\Lambda}) = p(\boldsymbol{\lambda}) \prod_{n} p\left(\mathbf{U}^{(n)}|\boldsymbol{\lambda}\right), \;\; p\left(\mathbf{U}^{(n)} \mid \boldsymbol{\lambda}\right) = \prod_{i,j} \mathcal{N}\left(u_{ij}^{(n)} \mid 0, \lambda_j\right). \tag{4.6}$$

  Here $u_{ij}^{(n)}$ is the (i,j)-th element of $\mathbf{U}^{(n)}$. Each entry of $\boldsymbol{\lambda}$ controls one column of each factor matrix. If a single entry λ_j approaches zero, then the prior mean and prior variance of $u_{ij}^{(n)}$ are both close to zero for all row indices $i \in [1, I_n]$ and mode indices $n \in [1, d]$. This encourages the whole j-th column of $\mathbf{U}^{(n)}$ to shrink to zero, leading to a rank reduction. The vector $\boldsymbol{\lambda}$ is shared across all modes, therefore it will shrink the same column of all CP factor matrices simultaneously, as shown in Fig. 4.2 (a).

- **Tucker Format:** A Tucker factorization includes a core tensor and d factor matrices, therefore $\mathbf{\Phi} = \{\boldsymbol{\mathcal{G}}, \{\mathbf{U}^{(n)}\}_{n=1}^{d}\}$. We also assign each factor matrix $\mathbf{U}^{(n)}$ with a variance-tunable Gaussian distribution. A Tucker model has d separate rank parameters $(r_1, \ldots, r_d)$ to determine, one per factor matrix as shown in Fig. 4.2 (b). Furthermore, the factor matrices and core tensor are handled separately. Therefore, we propose the following prior distributions:

$$p(\mathbf{\Phi}, \mathbf{\Lambda}) = p(\boldsymbol{\mathcal{G}}) \prod_{n} p\left(\mathbf{U}^{(n)}|\boldsymbol{\lambda}^{(n)}\right) p\left(\boldsymbol{\lambda}^{(n)}\right), \;\; p\left(\mathbf{U}^{(n)} \mid \boldsymbol{\lambda}^{(n)}\right) = \prod_{i,j} \mathcal{N}\left(u_{ij}^{(n)} \mid 0, \lambda_j^{(n)}\right).$$

  We use d independent rank controlling vectors $\{\boldsymbol{\lambda}^{(n)}\}_{n=1}^{d}$ to control the prior variances of different factor matrices separately. The j-th element of $\boldsymbol{\lambda}^{(n)}$ controls the j-th column of factor matrix $\mathbf{U}^{(n)}$. Therefore $\boldsymbol{\lambda}^{(n)}$ controls r_n, the n-th entry of the Tucker rank. We

place a weak normal prior over the entries of the core tensor $\boldsymbol{\mathcal{G}}$:

$$p\left(\boldsymbol{\mathcal{G}}\right) = \prod_{i_1,\ldots,i_d} \mathcal{N}\left(g_{i_1\ldots i_d} \mid 0, \sigma_0\right). \tag{4.7}$$

We make this choice to simplify parameter inference compared to the alternative of placing low-rank priors on both of the core tensor and the factor matrices.

- **Tensor-Train (TT) Format:** A TT factorization has d order-3 TT cores, therefore $\boldsymbol{\Phi} = \{\boldsymbol{\mathcal{G}}^{(n)}\}_{n=1}^{d}$. The TT format requires a more complicated prior because each TT core $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{r_{n-1}\times I_n \times r_n}$ depends on two rank parameters r_{n-1} and r_n. In order to automatically determine the TT rank, we choose $R_n > r_n$, and initialize the n-th TT core with size $R_{n-1} \times I_n \times R_n$. The prior density of all TT cores are given as

$$\begin{aligned} p(\boldsymbol{\Phi}, \boldsymbol{\Lambda}) &= p\left(\boldsymbol{\mathcal{G}}^{(d)} | \boldsymbol{\lambda}^{(d-1)}\right) \prod_{1\le n\le d-1} p\left(\boldsymbol{\mathcal{G}}^{(n)} | \boldsymbol{\lambda}^{(n)}\right) p\left(\boldsymbol{\lambda}^{(n)}\right), \\ p\left(\boldsymbol{\mathcal{G}}^{(n)} \mid \boldsymbol{\lambda}^{(n)}\right) &= \prod_{i,j,k} \mathcal{N}\left(g_{ijk}^{(n)} \mid 0, \lambda_k^{(n)}\right) \text{ for } n \in [1, d-1], \\ p\left(\boldsymbol{\mathcal{G}}^{(d)} \mid \boldsymbol{\lambda}^{(d-1)}\right) &= \prod_{i,j,k} \mathcal{N}\left(g_{ijk}^{(d)} \mid 0, \lambda_i^{(d-1)}\right). \end{aligned} \tag{4.8}$$

We introduce a vector $\boldsymbol{\lambda}^{(n)} \in \mathbb{R}^{R_n}$ to control the actual rank r_n for mode 1 to $d-1$. As shown in Fig. 4.2 (c), the k-th element of $\boldsymbol{\lambda}^{(n)}$ (i.e., $\lambda_k^{(n)}$) controls the prior variance of a slice $\boldsymbol{\mathcal{G}}^{(n)}(:,:,k)$. If $\lambda_k^{(n)}$ is small, the whole slice $\boldsymbol{\mathcal{G}}^{(n)}(:,:,k)$ is close to zero, leading to a rank reduction in the n-th mode. Parameter $\boldsymbol{\lambda}^{(d-1)}$ controls two separate cores. This prevents any rank parameters from overlapping and it simplifies posterior inference.

- **Tensor-Train Matrix (TTM) Format:** Similar to the TT format, a TTM decomposition also has d core tensors, therefore $\boldsymbol{\Phi} = \{\boldsymbol{\mathcal{G}}^{(n)}\}_{n=1}^{d}$. The only difference is that each $\boldsymbol{\mathcal{G}}^{(n)}$ is an order-4 tensor, which is initalized with a size $R_{n-1} \times I_n \times J_n \times R_n$ in our

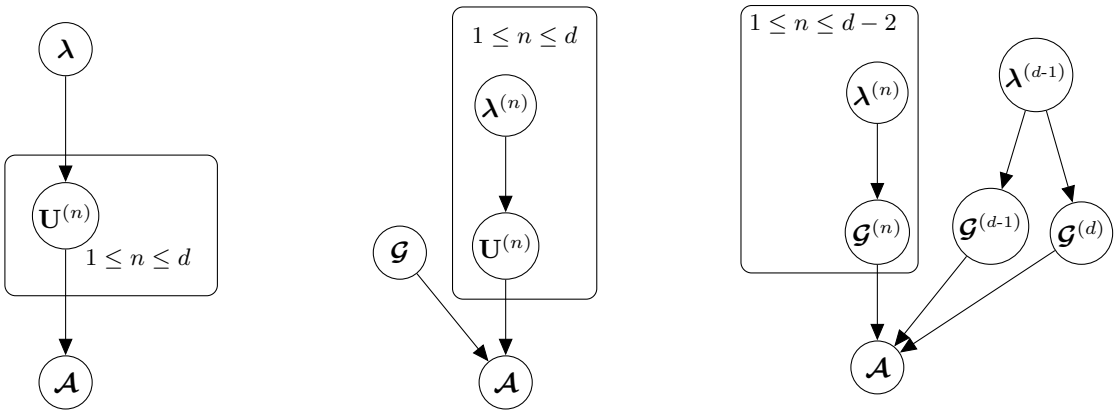


Figure 4.3: (a) CP graphical model (b) Tucker graphical model (c) TT/TTM graphical model.

Bayesian model. The prior for the TTM low-rank factors is

$$
\begin{aligned}
p(\boldsymbol{\Phi}, \boldsymbol{\Lambda}) &= p\left(\boldsymbol{\mathcal{G}}^{(d)} | \boldsymbol{\lambda}^{(d-1)}\right) \prod_{1 \le n \le d-1} p\left(\boldsymbol{\mathcal{G}}^{(n)} | \boldsymbol{\lambda}^{(n)}\right) p\left(\boldsymbol{\lambda}^{(n)}\right), \\
p\left(\boldsymbol{\mathcal{G}}^{(n)} \mid \boldsymbol{\lambda}^{(n)}\right) &= \prod_{i,j,k,l} \mathcal{N}\left(g_{ijkl}^{(n)} \mid 0, \lambda_l^{(n)}\right), \text{ for } n \in [1, d-1], \\
p\left(\boldsymbol{\mathcal{G}}^{(d)} \mid \boldsymbol{\lambda}^{(d-1)}\right) &= \prod_{i,j,k,l} \mathcal{N}\left(g_{ijkl}^{(d)} \mid 0, \lambda_i^{(d-1)}\right).
\end{aligned}
\tag{4.9}
$$

This prior very similar to that of TT format. We use a vector parameter $\boldsymbol{\lambda}^{(n)}$ to control the actual rank r_n of the n-th mode for $n \in [1, d-1]$, and $\boldsymbol{\lambda}^{(d-1)}$ is shared among $\boldsymbol{\mathcal{G}}^{(d)}$ and $\boldsymbol{\mathcal{G}}^{(d-1)}$.

4.1.3 Rank-Shrinking Hyper-Parameter Priors

To complete the setup of the full Bayesian model (4.1), we still need to specify the prior of rank-control hyper parameters $\boldsymbol{\Lambda} = \boldsymbol{\lambda}$ (for CP) or $\boldsymbol{\Lambda} = \{\boldsymbol{\lambda}^{(n)}\}_{n=1}^{d}$ (for Tucker, TT and TTM). Since small elements in $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^{(n)}$ lead to rank reductions in the tensor models, we choose two hyper-prior densities that place high probability near zero. We focus our notation in this subsection on the CP model for simplicity.

We consider two choices of prior on the hyper parameter $\boldsymbol{\lambda}$: the Half-Cauchy with scale parameter η and the improper Log-Uniform on $(0, \infty)$:

$$p(\boldsymbol{\lambda}) = \prod_{i=1}^{R} p(\lambda_i), \quad \text{with } p(\lambda_i) = \begin{cases} \text{HC}(\sqrt{\lambda_i}|0, \eta) \text{ or} \\ \text{LU}(\sqrt{\lambda_i}). \end{cases} \tag{4.10}$$

The improper Log-uniform distribution has a fatter tail than the Half-Cauchy distribution and is parameter-free. We illustrate both densities in Fig. 4.4a. The Half-Cauchy scaling parameter $\eta > 0$ can be adjusted to tune the tradeoff between accuracy and rank-sparsity. Decreasing the magnitude of η increases rank-sparsity. Both the Half-Cauchy density function

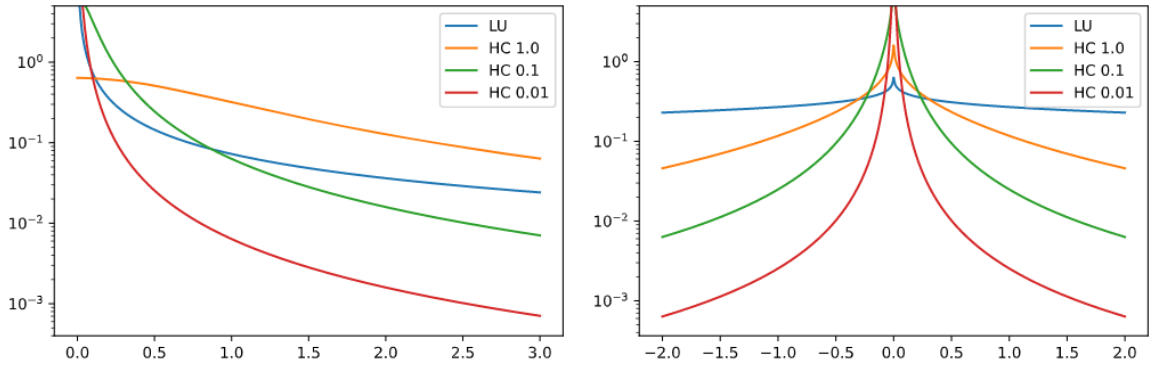
$$\text{HC}(x|0, \eta) \propto \left(1 + \frac{x^2}{\eta^2}\right)^{-1} \tag{4.11}$$

and the Log-Uniform density function

$$\text{LU}(x) \propto x^{-1} \tag{4.12}$$

place high probability in regions around zero. The parameter $\boldsymbol{\lambda}$ controls the prior variance of the tensor factors in $\boldsymbol{\Phi}$, all of which have prior mean zero. Therefore the prior density encodes a prior belief that the tensor rank is low, and it encourages structured rank shrinkage. We provide the Bayesian graphical models for each low-rank tensor format in Fig. 4.3.

In Fig 4.4 we demonstrate how our prior induces rank-sparsity in a CP model. Fig 4.4a plots the prior density on the rank parameter λ_j. Fig. 4.4b shows the corresponding marginal prior on $u_{ij}^{(n)}$. The flat tail and sharp peak of the marginal prior induced by the Log-Uniform rank hyper-prior leads to strong shrinkage of small values of $u_{ij}^{(n)}$ towards 0 but permits medium values to escape the "gravitational pull" around 0 [106]. In comparison, the marginal Horseshoe prior induced by the Half-Cauchy hyper-prior exerts a weaker shrinkage effect at small values of $u_{ij}^{(n)}$ but a stronger shrinkage effect on larger values.



(a) (b)

Figure 4.4: (a) Comparison of the probability density functions of the Log-Uniform and Half-Cauchy hyperprior on λ_j. Several values of the Half-Cauchy scale parameter η are given. (b) Comparison of the probability density functions of the corresponding marginal prior on the low-rank tensor factor entry $u_{ij}^{(n)}$.

4.2 Scalable Parameter Inference

Now we discuss how to estimate the resulting posterior density (4.1). We develop an efficient tensorized Bayesian inference approach by improving Stochastic Variational Inference (SVI) [69]. We consider SVI [69] due to its superior computational and memory efficiency over gradient-based MCMC [66] and Stein variational gradient descent [88]. However, directly applying SVI to our tensorized training can cause numerical failures. Therefore, we will develop a customized SVI solver with analytical/numerical hybrid parameter update that is suitable for our Bayesian tensorized neural networks.

4.2.1 Challenges in Training Bayesian Tensorized Neural Networks

Now we explain the challenges of directly applying SVI to train our Bayesian tensorized neural network model. As an example, we focus our notation on the CP-format one-layer model with parameters

$$\boldsymbol{\theta} = \{\boldsymbol{\Phi}, \boldsymbol{\Lambda}\} = \{\{\mathbf{U}^{(n)}\}_{n=1}^{d}, \boldsymbol{\lambda}\}. \tag{4.13}$$

The extension to other tensor formats and to multiple layers is trivial. For notational convenience we omit the description of the bias term $\mathbf{b}$ since it is assigned a Normal variational posterior and follows the standard update rules specified in [70].

In variational inference, it is a common practice to simplify a posterior density in order to reduce the computational cost. In our problem setting, we firstly use the mean-field approximation [107] to achieve a tractable optimization:

$$q\left(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda}\right) = q_{\mathbf{U}}\left(\{\mathbf{U}^{(n)}\}\right) q_{\boldsymbol{\lambda}}\left(\boldsymbol{\lambda}\right). \tag{4.14}$$

We further model the posterior of the tensor factors with a normal distribution

$$q_{\mathbf{U}}\left(\{\mathbf{U}^{(n)}\}\right) = \prod_{n=1}^{d} q_{\mathbf{U}^{(n)}}\left(\mathbf{U}^{(n)}\right), \;\; q_{\mathbf{U}^{(n)}}\left(\mathbf{U}^{(n)}\right) = \prod_{i,j} \mathcal{N}\left(u_{ij}^{(n)} | \overline{u_{ij}^{(n)}}, {\Sigma_{ij}^{(n)}}^2\right), \tag{4.15}$$

where $\overline{u_{ij}^{(n)}}$ and $\Sigma_{ij}^{(n)}$ are the (i,j)-th elements of the unknown posterior mean $\overline{\mathbf{U}^{(n)}}$ and posterior standard deviation $\boldsymbol{\Sigma}^{(n)}$ to be inferred, respectively.

Now we discuss the challenges in learning the variational posterior distribution. We modify Eq. (2.17) to obtain our objective function:

$$\mathcal{L}\left(q\right) = -\mathbb{E}_{q\left(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda}\right)} \log\, p(\mathcal{D}|\{\mathbf{U}^{(n)}\}) + \mathrm{KL}\left(q\left(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda}\right) || p(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda})\right). \tag{4.16}$$

Due to the nonlinear tensorized forward model, we need to employ gradient-based iterations in SVI to update the tensor factor parameters. The expected log-likelihood in Equation (4.16) must be approximated by sampling the variational distribution q. The first standard approach is to select a variational distribution $q\left(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda}\right)$ for which the KL divergence in Equation (4.16) can be obtained in a closed form. The second standard approach is to approximate the KL divergence term by sampling from the variational posterior. In practice, two challenges prevent us from applying these standard SVI approaches:

- **Challenge 1: Closed-form objectives require multiple training runs:** Variational

distributions q that permit a closed-form approximation of the KL divergence require additional hyperparameters. Existing distributions that enable a closed-form KL divergence require a hierarchical Bayesian parameterization of the rank parameter $\boldsymbol{\lambda}$ [108,109], requiring up to five additional hyperparameters for the new random variables [108]. Additional hyperparameters would require additional tuning runs and remove the benefits of one-shot tensorized training. Therefore, we avoid this option.

- **Challenge 2: Sampling-based approximation increases gradient variance:** Sampling-based approximation of the KL divergence leads to gradient instability during rank shrinkage. The gradient variance with respect to the low-rank tensor factor parameters is proportional to the variance of $1/\boldsymbol{\lambda}$, and it may explode during rank-shrinkage as $\boldsymbol{\lambda}$ approaches 0, so sampling $\boldsymbol{\lambda}$ is not feasible.

We provide more details about the second challenge. We consider the gradient of the objective function in Eq. 4.16 w.r.t. the parameters $\overline{u_{ij}^{(n)}}$ and $\Sigma_{ij}^{(n)}$. First we observe that

$$\mathrm{KL}\left(\mathcal{N}\left(u_{ij}^{(n)}|\overline{u_{ij}^{(n)}}, {\Sigma_{ij}^{(n)}}^2\right)||\mathcal{N}\left(u_{ij}^{(n)}|0,\lambda_j\right)\right) \propto \frac{\overline{u_{ij}^{(n)}}^2 + {\Sigma_{ij}^{(n)}}^2}{\lambda_j}. \tag{4.17}$$

Let ϕ represent either parameter of $\left\{\overline{u_{ij}^{(n)}}, \Sigma_{ij}^{(n)}\right\}$. Then sampling $\boldsymbol{\lambda}$ yields a gradient variance

$$\mathbb{V}\left[\nabla_\phi \mathrm{KL}\left(\mathcal{N}\left(u_{ij}^{(n)}|\overline{u_{ij}^{(n)}}, {\Sigma_{ij}^{(n)}}^2\right)||\mathcal{N}\left(u_{ij}^{(n)}|0,\lambda_j\right)\right)\right] \propto \mathbb{V}\left[\frac{1}{\lambda_j}\right]. \tag{4.18}$$

The goal of our low-rank prior is to shrink many λ_j's to 0 in the training process. If the distribution of λ_j is non-degenerate, even small uncertainties in the value of λ_j will lead to large variance in Equation (4.18) as the posterior probability of λ_j concentrates around 0 . As a result, a rank shrinkage can cause high-variance gradients which in turn may increase the magnitude of factor matrix parameters, as shown in Fig. 4.5.

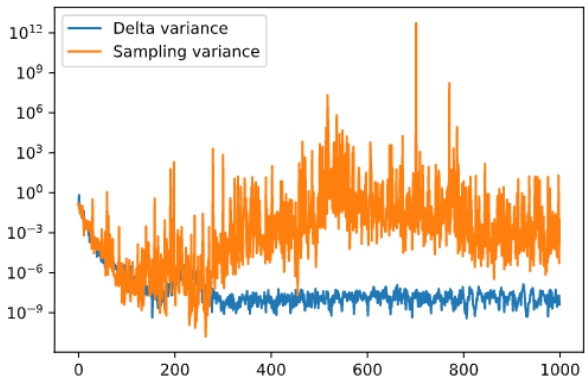


Figure 4.5: The gradient variance of a single low-rank tensor factor parameter. Sampling the rank parameter $\boldsymbol{\lambda}$ leads to high-variance gradients, while our proposed delta approximation of hyper parameters reduces the gradient variance significantly (see Section 4.2.2 and Section 4.2.3).

4.2.2 Simplified Posterior for Rank-Controlling Hyper Parameters

To avoid gradient variance explosion, we propose a deterministic approximation to the hyper parameter $\boldsymbol{\lambda}$:

$$q_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = \delta_{\overline{\boldsymbol{\lambda}}}(\boldsymbol{\lambda}) \tag{4.19}$$

where δ is a Delta function and $\overline{\boldsymbol{\lambda}}$ is the posterior mean of $\boldsymbol{\lambda}$. This delta approximation was used for empirical partially Bayes estimation in [110]. This approximation admits *closed-form updates* to the following sub-problem when the factor matrices are fixed:

$$\arg\min_{\overline{\lambda}_k} \mathrm{KL}\left(q\left(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda}\right) || p(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda})\right). \tag{4.20}$$

We provide the closed-form analytical updates for $\overline{\lambda}_k$ under each choice of prior in CP format and give the details in Appendix A. The results associated with other tensor formats can be obtained similarly. For the Log-Uniform prior

$$\overline{\lambda}_k^{\star} \leftarrow \frac{M}{D+1}. \tag{4.21}$$

Here we have used the notations

$$D = \sum_n I_n, \quad M = \sum_{1 \le n \le d} \sum_{1 \le i \le I_n} \overline{u_{ik}^{(n)}}^2 + \Sigma_{ik}^{(n)^2}. \tag{4.22}$$

The number of entries controlled by λ_j is D, and M is their combined magnitude and variance. In the case of the Half-Cauchy prior with scale parameter η, the update is

$$\overline{\lambda_k^\star} \leftarrow \frac{M - \eta^2 D + \sqrt{M^2 + (2D+8)\eta^2 M + \eta^4 D^2}}{2D+2}. \tag{4.23}$$

For the Half-Cauchy hyperprior, decreasing the magnitude of the scale parameter η decreases the magnitude of the update of $\overline{\lambda_k^\star}$, thereby increasing rank-sparsity.

4.2.3 Analytical/Numerical Hybrid Parameter Update in SVI

With the proposed Delta posterior approximation for $\boldsymbol{\lambda}$, now we can train our tensorized neural network training with an analytical/numerical hybrid parameter update rule in SVI. Specifically, in every iteration of SVI, we use a gradient-based half step to update the tensor factors in $\boldsymbol{\Phi}$ and closed-form half step to the hyper parameters $\overline{\boldsymbol{\lambda}}$. We apply the reparametrization trick

$$u_{ij}^{(n)} = \overline{u_{ij}^{(n)}} + z\Sigma_{ij}^{(n)}, \qquad z \sim \mathcal{N}(0,1) \tag{4.24}$$

to sample from the tensor factor distributions.

- **Half Step 1: Gradient Update for tensor factors:** We sample the low-rank tensor factors $\boldsymbol{\Phi}$ and update all parameters of the tensor factor variational distributions using gradient descent on the loss $\mathcal{L}(q)$ of Eq. (4.16) with a learning rate α:

$$\boldsymbol{\Phi} \leftarrow \boldsymbol{\Phi} + \alpha \nabla_{\boldsymbol{\Phi}} \mathcal{L}(q). \tag{4.25}$$

In the the CP model, the gradients for the posterior variance and mean of the factor matrices are given by

$$\begin{aligned}\nabla_{\Sigma_{ij}^{(n)}}\mathcal{L}(q) &= -z\nabla_{u_{ij}}\log\, p(\mathcal{D}|\{\mathbf{U}^{(n)}\}) - \frac{1}{\Sigma_{ij}^{(n)}} + \frac{\Sigma_{ij}^{(n)}}{\overline{\lambda}_j} \\ \nabla_{\overline{u_{ij}^{(n)}}}\mathcal{L}(q) &= -\nabla_{u_{ij}}\log\, p(\mathcal{D}|\{\mathbf{U}^{(n)}\}) + \frac{\overline{u_{ij}^{(n)}}}{\overline{\lambda}_j}.\end{aligned} \tag{4.26}$$

Note that z is the random variable sampled during the forward pass due to the reparameterization in Eq. (4.24) and the gradients with respect the log-likelihood are computed using standard automatic differentiation. We describe the gradients for the other three tensor formats in Appendix B.

- **Half Step 2: Incremental closed-form update for $\overline{\boldsymbol{\lambda}}$:** We analytically update the rank-controlling parameters $\boldsymbol{\lambda}$ based on the results in (4.21) and (4.23). We found empirically that incremental updates, rather than direct assignment of the results from (4.21) or (4.23), led to better performance. Therefore we adopt an incremental update strategy with learning rate γ for the rank parameter updates:

$$\overline{\lambda}_k \leftarrow \gamma\overline{\lambda}_k^{\star} + (1-\gamma)\overline{\lambda}_k. \tag{4.27}$$

As shown in Figure 4.5, this proposed hybrid parameter update can greatly reduce the gradient variance of tensor factors.

4.2.4 Algorithm Flow and Implementation Issues

The full description of our end-to-end tensorized training with rank determination is shown in Alg. 1. We iteratively repeat the hybrid parameter updates for a predetermined number of epochs m. In the following, we discuss some important implementation issues.

Algorithm 1 SVI-Based Tensorized Training with Rank Determination

Input: Factor learning rate α, EM stepsize γ, rank cutoff ϵ, warmup epochs e_w, total epochs m, tuning epochs t
for Epoch e in $[1, \ldots, m]$ **do**
  Assign β according to Equation (4.28).
  for each batch $\mathcal{B} \subset \mathcal{D}$ **do**
    Update the low-rank factor distribution variational parameters as in Half Step 1, Equation (4.25).
    Update the rank-control hyper-parameters as in Half Step 2, Equation (4.27).
  end for
end for
Prune tensor ranks as described in Equation (4.30).

Warmup Schedule A general challenge in Bayesian tensor computation is that poor initializations can lead to excessive rank shrinkage and trivial rank-zero solutions. In linear tensor problems such as tensor completion the SVD is used to generate high-quality initializations [58, 62]. For nonlinear tensorized neural networks we randomly initialize the factor matrices so the predictive accuracy is low and the KL divergence to the prior may dominate the local loss landscape around the initialization point. To avoid trivial rank-zero local optima early in the training process, we incrementally re-weight the KL divergence from the variational approximation to the prior during the training process. Let e_w be the number of warmup training epochs and e be the current epoch. We re-weight the KL divergence from the variational approximation to the prior by a factor β defined by

$$\beta = \min\left(1, \frac{e}{e_w}\right), \tag{4.28}$$

and update the loss from Eq. (4.16) accordingly:

$$\mathcal{L}(q) = -\log \mathbb{E}_{q(\{\mathbf{U}^{(n)}\},\boldsymbol{\lambda})} p(\mathcal{D}|\{\mathbf{U}^{(n)}\}) + \beta \mathrm{KL}\left(q\left(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda}\right) || p(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda})\right). \tag{4.29}$$

Gradually increasing the weight of the KL divergence to the prior avoids early local optima in which all ranks shrink to zero. We have found empirically that $e_w = m/2$ is a good choice for the number of warmup steps.

Table 4.1: Summary of different training methods.

Method	memory cost of training	# training runs	model size for inference
Baseline	high	1	huge
FR [50]	low	many	small
TC-MR [47]	high	1	small
TC-OR [47]	high	1	small
ARD-LU (Proposed)	low	1	small
ARD-HC (Proposed)	low	1	small

Rank Pruning After we run our Bayesian solver we truncate the ranks with variance $\overline{\lambda}_k$ below a pre-specified threshold ϵ. For example, for the CP format if $\overline{\lambda}_k < \epsilon$ we assign

$$\overline{u_{ik}^{(n)}} \leftarrow 0 \text{ and } \Sigma_{ik}^{(n)} \leftarrow 0 \text{ for } 1 \leq n \leq d, 1 \leq i \leq I_n. \tag{4.30}$$

The associated k-th column of $\mathbf{U}^{(n)}$ is removed, leading to a rank shrinkage and automatic model parameter reduction.

4.3 Experiments

We demonstrate the applications of our rank-adaptive tensorized end-to-end training method on several neural network models. Our method trains a Bayesian neural network, therefore we report the predictive accuracy of the posterior mean. In order to compare the performance, we implement the following methods in our experiments:

- **Baseline:** a standard training method, where model parameters are uncompressed.
- **TC-MR** [47, 48]: train and then compress with maximum ranks. We train a uncompressed neural network with the "baseline" method, followed by a tensor decomposition and fine-tuning. For the DLRM model we fine-tune for one epoch. In all other experiments we fine-tune for 20 epochs. This approach requires that the user select the compression rank. Here we use the maximum rank used in our Bayesian model. This

approach has been studied for computer vision tasks using the CP decomposition in [47] and the Tucker decomposition in [48, 89]. We compare against the algorithms of [47, 48], but on different architectures.

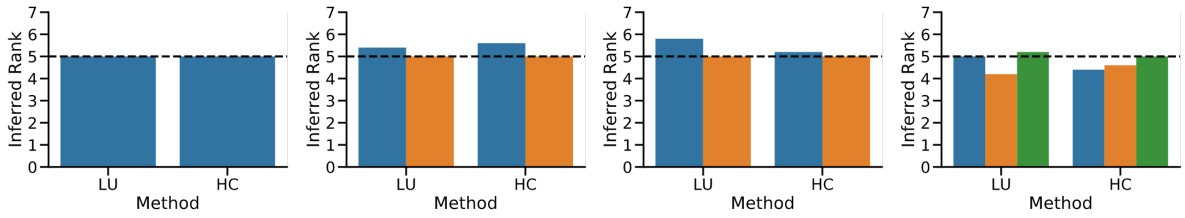
- **TC-OR**: train and then compress with oracle rank (r in CP or $\mathbf{r} = [r_1, r_2, \cdots, r_d]$ for other formats). This method follows the same procedures of TC-MR [47, 48], except that it uses the "oracle rank" discovered by our proposed rank determination method. In practice this "TC-OR" method would require a combinatorial rank search over a high-dimensional discrete space to discover the same rank as our method.

- **FR**: Fixed-rank tensorized training. We implement tensorized training [50, 51, 52, 103] with a tensor rank fixed *a priori*. Determining the tensor ranks is challenging in this approach. In our experiments we reuse the well-tuned parameters from previous literature. The CNN experiment and architecture in the supplemental material is taken from [51]. The NLP and DLRM experiment architectures are taken from [103].

- **ARD-LU:** the first version of our proposed tensorized training method with automatic rank determination. We use the log-uniform prior in (4.12) for the rank-control hyper-parameters. All tensor factors are initialized with a maximum rank (R for CP and $\mathbf{R} = [R, \cdots, R_d]$ for other formats), and the actual ranks (r for CP and $\mathbf{r} = [r_1, \cdots r_d]$ for other formats) are automatically determined by our training process. To compare our method with FR, we set the maximum rank to the rank used in FR.

- **ARD-HC:** the second version of our proposed training method using the half-Cauchy prior (4.11) for the rank-control hyper parameters.

As shown in Table 4.1 our proposed methods enjoy all of the listed advantages compared with other methods. The proposed automatic tensor rank determination avoids the expensive multiple training runs in FR, and it also results in the (almost) smallest models for inference. We consider four low-rank tensor formats for each tensorized method. Therefore, our experiments



(a) CP format (b) TT format (c) TTM format (d) Tucker format

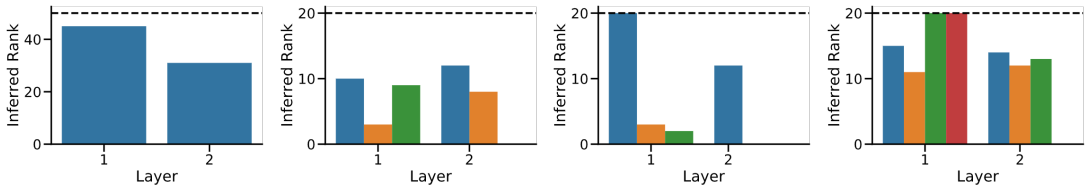
Figure 4.6: The inferred ranks for a synthetic example. The true rank (dashed lines) is 5 and maximum rank is set to 10. The inferred ranks of different modes are given by colored bars.

involve the implementation of 21 specific methods in total (20 tensorized implementations plus one baseline method). For all experiments we list the full tensor dimension and rank settings in the supplement. For all experiments we set the rank parameter learning rate $\gamma = 0.9$.

In our Bayesian training, every tensorized model parameter is equipped with two training variables (i.e., posterior mean and variance). Therefore the number of training variables is $2\times$ that of the tensorized model parameter numbers. This parameter overhead in Bayesian training brings in the capability of uncertainty quantification in output prediction, which is important for safety-critical applications. Our Bayesian model also allows a point-wise maximum-a-posterior (MAP) training. In MAP training, the only additional parameters required are the rank-control parameters so the number of training variables is only slightly larger than the number of training variables in fixed-rank tensorized training.

4.3.1 Synthetic Example for Rank Determination

First we test the ability of our proposed method to infer the tensor rank of model parameters in a neural network. For each tensor format we construct a synthetic version of the MNIST dataset using a one-layer tensorized neural network (equivalent to tensorized logistic regression). The tensorized layer is fully connected and the fixed tensor rank is five for each tensor format: 5 for CP, [5, 5, 5] for Tucker and [1, 5, 5, 1] for TT/TTM. We use the rank-5 model to generate synthetic labels for the MNIST images. Then we train a set of low-rank tensorized models with a maximum rank of 10 on the synthetic dataset. For the CP, tensor-train, and Tucker



(a) CP format (b) TT format (c) TTM format (d) Tucker format

Figure 4.7: Inferred ranks for one run of the MNIST experiment using a log-uniform prior. The maximum rank is given by a dashed black line. The inferred ranks are given by colored bars.

Table 4.2: Tensorization settings for the MNIST example.

Model	Layer 1 Dimensions	Layer 2 Dimensions	Max Rank
Baseline	784×512	512×10	NA
CP	[28,28,16,32]	[32,16,10]	50
Tucker	[28,28,16,32]	[32,16,10]	20
TT	[28,28,16,32]	[32,16,10]	20
TTM	[4,7,4,7],[4,4,8,4]	[32,16],[2,5]	20

formats we reshape the weight matrix $\mathbf{W} \in \mathbb{R}^{784\times10}$ into a tensor of shape size $[28, 28, 10]$ (i.e., an order-3 tensor of size $28 \times 28 \times 10$). For the tensor-train matrix format we use the dimensions $[4, 7, 4], [7, 2, 5]$.

We plot the mean inferred ranks for our log-uniform (LU) and half-cauchy (HC) priors in Fig. 4.6. The actual CP rank is exactly recovered in our model. The inferred ranks of Tucker, TT and TTM are close to but not equal to the exact values, because tensor ranks are not unique, which is a fundamental difference between matrices and tensors.

4.3.2 MNIST

Next we test a neural network with two fully connected (FC) layers on the MNIST dataset with images of size 28×28. The first FC layer is size 784×512 and has a ReLU activation function. The second FC layer is size 512×10 with a softmax activation function. Exact tensor dimensions are given in Table 4.2. In all cases our automatic rank determination can achieve the highest compression ratio in training. Our proposed automatic rank determination both improves accuracy and reduces parameter number in all tensor formats except the TT

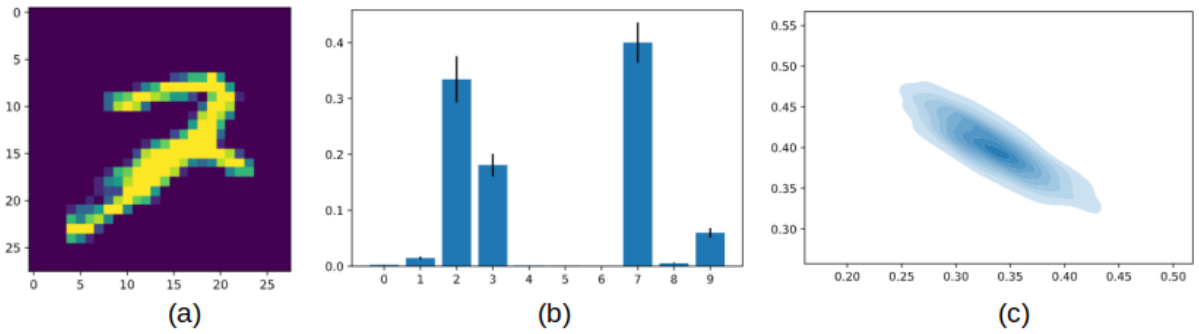


Figure 4.8: (a) A challenging MNIST image with true label "2". (b) Mean and standard deviation of the CP ARD-LU model softmax outputs. (c) Marginal predictive density of the two most likely labels "2" (x-axis) and "7" (y-axis).

format which has slight accuracy loss but the highest compression ratio. We hypothesize that the automatic rank reduction can reduce over-fitting on the simple MNIST task. The TTM format is best-suited to fully connected layers, achieving the second-highest compression ratios and the second-best accuracy. In Fig. 4.7 we plot the rank determination output of a single training run using our log-uniform prior. We note that our algorithm discovers the actual ranks that are nearly impossible to determine via hand-tuning or combinatorial search (for example [1,20,3,2,1] in the TTM model from a maximum rank of [1,20,20,20,1], which may require up to 16,000 searches).

With the obtained Bayesian solution, we can quantify the uncertainty of our model as a by-product. Popular metrics for uncertainty measures include negative log-likelihood, expected calibration error, which measures model over-/under-confidence, and out-of-distribution input detection [111]. In Fig. 4.8, we show the classification uncertainty of an image that is hard to recognize in practice. With the CP tensorized model trained from ARD-LU, we plot the mean and variance of the predicted softmax outputs in Fig. 4.8 (b). This plot clearly shows that this image looks like "2", "3" or "7", with the highest probability of being classified as "7". Fig. 4.8 (c) further plots the marginal predictive density of the two most likely labels "2" and "7".

Table 4.3: Training results of the MNIST example.

Tensor Type	Model	Training Param. #	Final Param. #	Accuracy
	Baseline	407,050	407,050	98.09
CP	FR	8,622 (47.2×)	8,622 (47.2×)	97.52
	TC-MR [47]	407,050 (1×)	8,622 (47.2×)	97.32
	TC-OR [47]	407,050 (1×)	7,175 (56.7×)	97.36
	ARD-LU (Proposed)	17,344 (23.5×)	7,175 (56.7×)	98.06
	ARD-HC (Proposed)	17,344 (23.5×)	7,134 (57.1×)	97.98
Tucker	FR [52]	171,762 (2.4×)	171,762 (2.4×)	97.93
	TC-MR [48]	407,050 (1×)	171,762 (2.4×)	98.00
	TC-OR [48]	407,050 (1×)	100,758 (4.0×)	97.91
	ARD-LU (Proposed)	343,644 (1.18×)	100,758 (4.0×)	98.30
	ARD-HC (Proposed)	343,644 (1.18×)	91,332 (4.5×)	98.30
TT	FR [50]	26,562 (13.9×)	26,562 (15.3×)	97.78
	TC-MR	407,050 (1×)	26,562 (15.3×)	97.43
	TC-OR	407,050 (1×)	4,224 (96.4×)	96.91
	ARD-LU (Proposed)	53,224 (7.65×)	4,224 (96.4×)	96.28
	ARD-HC (Proposed)	53,224 (7.65×)	4,276 (95.2×)	97.04
TTM	FR [50]	29,242 (13.9×)	29,242 (13.9×)	98.06
	TC-MR	407,050 (1×)	29,242 (13.9×)	97.47
	TC-OR	407,050 (1×)	6,144 (66.3×)	96.61
	ARD-LU (Proposed)	58,564 (6.95×)	6,144 (66.3×)	98.24
	ARD-HC (Proposed)	58,564 (6.95×)	5,200 (78.3×)	98.23

Note: the training parameters in ARD-LU and ARD-HC include posterior mean and variance, so the training parameter number is 2× of that in FR. The results of FR rely on manual rank tuning in contrast to our automatic rank determination procedure.

4.3.3 Embedding Table for Natural Language Processing (NLP)

We continue to validate our algorithm with a sentiment classification task from [103]. Like many NLP models, the first layer is a large embedding table. Embedding tables are a promising target for tensor compression because their required input dimension equals the number of unique tokens in the input dataset (i.e. number of vocabulary words, number of users). Tensor decomposition can enforce weight sharing and dramatically reduce the parameter count of these models. Recent work in tensorized neural networks has applied the TTM format to compress large embedding tables with a high ratio [103]. We replicate a sentiment classification model on the IMDB dataset from their work. The neural network model consists of an embedding table

Table 4.4: Training results on the NLP embedding table.

Tensor Type	Model	Training Parameter #	Final model Parameter #	Accuracy
	Baseline	6,400,000	6,400,000	88.34
CP	FR	8,276 (774×)	8,276 (774×)	87.44
	TC-MR	6,400,000 (1×)	8,276 (774×)	74.46
	TC-OR	6,400,000 (1×)	6,138 (1024×)	73.21
	ARD-LU (Proposed)	16,602 (385×)	6,138 (1024×)	87.61
	ARD-HC (Proposed)	16,602 (385×)	6,476 (998×)	87.54
Tucker	FR	78,540 (81×)	78,540 (81×)	87.80
	TC-MR	6,400,000 (1×)	78,540 (81×)	75.12
	TC-OR	6,400,000 (1×)	61,920 (103×)	71.97
	ARD-LU (Proposed)	157,105 (40×)	61,920 (103×)	87.79
	ARD-HC (Proposed)	157,105 (40×)	58,120 (110×)	88.01
TT	FR [103]	28,260 (226×)	28,260 (226×)	85.6
	TC-MR	6,400,000 (1×)	28,260 (226×)	82.34
	TC-OR	6,400,000 (1×)	22,982 (278×)	71.81
	ARD-LU (Proposed)	56,640 (113×)	22,982 (278×)	85.33
	ARD-HC (Proposed)	56,640 (113×)	19,363 (331×)	85.82
TTM	FR [103]	22,312 (287×)	22,312 (287×)	88.59
	TC-MR	6,400,000 (1×)	22,312 (287×)	83.79
	TC-OR	6,400,000 (1×)	15,932 (402×)	84.83
	ARD-LU (Proposed)	44,724 (143×)	15,932 (402×)	88.93
	ARD-HC (Proposed)	44,724 (143×)	14,275 (448×)	88.78

Note: the training parameters in ARD-LU and ARD-HC include posterior mean and variance of each tensorized model parameter. The results of FR rely on manual rank tuning in contrast to our automatic rank determination procedure.

with dimension $25,000 \times 256$, two bidirectional LSTM layers with hidden unit size 128, and a fully-connected layer with 256 hidden units. Following the setting in [103] we do not tensorize these layers. Dropout masks are applied to the output of each layer except the last. Exact tensor dimensions are given in Table 4.5.

We test all methods on the sentiment classification problem. The tensor dimensions and maximum ranks used to compress the embedding table are given in the supplementary material. The outcomes of our experiments are reported in Table 4.4. Compared with all other tensor approaches, our methods (ARD-LU and ARD-HC) have achieved the best compression ratio for all tensor formats at little to no accuracy cost. The TTM format outperforms all other

Table 4.5: Tensorization settings for the NLP embedding table.

Model	Embedding Dimensions	Max Rank
Baseline	$25,000 \times 256$	NA
CP	[5,8,25,25,4,8,8]	50
Tucker	[25,25,40,16,16]	5
TT	[5,8,25,25,4,8,8]	20
TTM	[5,5,5,5,6,8],[2,2,2,2,4,4]	20

models (including the baseline uncompressed model) in terms of accuracy, though we note that the CP model performs well despite its extremely low parameter number.

4.3.4 Deep Learning Recommendation System (DLRM)

We continue to use our proposed Bayesian tensorized method to train the benchmark Deep Learning Recommendation Model (DLRM) [3]. In the DLRM model, embedding tables are used to process categorical features, while continuous features are processed with a bottom multilayer perceptron (MLP). Then, second-order interactions of different features are computed explicitly. The results are processed with a top MLP and fed into a sigmoid function in order to give a probability of a click. The whole model has over 4 billion training variables.

We tensorize the five largest embedding tables to reduce the training variables. Exact tensor dimensions are given in Table 4.6. Our experiment results are reported in Table 4.7. Our proposed automatic rank reduction enables parameter reduction at little to no accuracy cost over fixed-rank tensorized training. Our approach outperforms the train-then-compress approach which requires expensive full-model training. Compared with baseline full-size training, our method achieves to up to $27,664\times$ (in TT format) parameter reduction during training with little accuracy loss. Our one-shot training also greatly increases the compression ratio over fixed-rank training at little to no accuracy cost, enabling up to $7\times$ higher compression ratios in the TTM model.

The train-then-compress approach can be expensive for this large-scale problem. Because

Table 4.6: Tensorization settings for the DLRM embedding tables.

Embedding Layer	Model	Embedding Dimensions	Max Rank
1	Baseline	$10,131,227 \times 128$	NA
	CP	[200,220,250,128]	350
	Tucker	[200,220,250,128]	20
	TT	[200,220,250,128]	24
	TTM	[200,220,250],[4,4,8]	16
2	Baseline	$2,202,608 \times 128$	NA
	CP	[125,130,136,128]	306
	Tucker	[125,130,136,128]	20
	TT	[125,130,136,128]	24
	TTM	[125,130,136],[4,4,8]	16
3	Baseline	$8,351,593 \times 128$	NA
	CP	[200,200,209,128]	333
	Tucker	[200,200,209,128]	22
	TT	[200,200,209,128]	24
	TTM	[200,220,250],[4,4,8]	16
4	Baseline	$5,461,306 \times 128$	NA
	CP	[166,175,188,128]	326
	Tucker	[166,175,188,128]	21
	TT	[166,175,188,128]	24
	TTM	[166,175,188],[4,4,8]	16
5	Baseline	$7,046,547 \times 128$	NA
	CP	[200,200,200]	335
	Tucker	[200,200,200,128]	22
	TT	[200,200,200,128]	24
	TTM	[200,200,200],[4,4,8]	16

Table 4.7: Training results on the DLRM embedding tables.

Tensor Type	Model	Training Parameter #	Final model Parameter #	Accuracy
	Baseline	4,248,739,968	4,248,739,968	78.75
CP	FR	1,141,597 (3,721×)	1,141,597 (3,721×)	78.60
	TC-MR	4,248,739,968 (1×)	1,141,597 (3,721×)	75.41
	TC-OR	4,248,739,968 (1×)	563,839 (7,535×)	74.92
	ARD-LU (Proposed)	2,284,844 (1860×)	563,839 (7,535×)	78.61
	ARD-HC (Proposed)	2,284,844 (1860×)	570,685 (7,444×)	78.57
Tucker	FR	1,131,212 (3,755×)	1,131,212 (3,755×)	78.60
	TC-MR	4,248,739,968 (1×)	1,131,212 (3,755×)	78.67
	TC-OR	4,248,739,968 (1×)	436,579 (9,731×)	78.50
	ARD-LU (Proposed)	2,262,852 (1,877×)	436,579 (9,731×)	78.64
	ARD-HC (Proposed)	2,262,852 (1,877×)	402,023 (10,568×)	78.62
TT	FR [103]	1,135,752 (3,740×)	1,135,752 (3,740×)	78.68
	TC-MR	4,248,739,968 (1×)	1,135,752 (3,740×)	78.39
	TC-OR	4,248,739,968 (1×)	153,582 (27,664×)	78.45
	ARD-LU (Proposed)	2,271,864 (1870×)	153,582 (27,664×)	78.67
	ARD-HC (Proposed)	2,271,864 (1870×)	159,529 (26,633×)	78.63
TTM	FR [103]	1,130,048 (3759×)	1,130,048 (3759×)	78.73
	TC-MR	4,248,739,968 (1×)	1,130,048 (3759×)	78.43
	TC-OR	4,248,739,968 (1×)	199,504 (21, 296×)	78.62
	ARD-LU (Proposed)	2,260,256 (1879×)	199,504 (21,296×)	78.72
	ARD-HC (Proposed)	2,260,256 (1879×)	163,976 (25,910×)	78.73

Note: the training parameters in ARD-LU and ARD-HC include posterior mean and variance of every tensorized model parameters, so the number of training variables is 2× of that in fixed-rank tensorized training (FR).The results of FR rely on manual rank tuning in contrast to our automatic rank determination procedure.

the trained embedding tables are extremely large, compressing them in Tucker or CP format is computationally expensive and time-consuming. This challenge can be avoided in our end-to-end-training approaches because we do not need to explicitly form the embedding tables.

4.3.5 CIFAR-10 Convolutional Model

Finally, we provide experiments on a convolutional neural network taken from [51]. This model consists of six convolutional layers followed by three fully connected layers. We follow [51] and tensorize all layers except the first convolution and the last fully connected layer

which together contain a small fraction of the total parameters. As before, we test all four tensor formats with our rank determination approach. The results of our method, the baseline model, and the train-and-then-compress approach are reported in Table 4.8. We use the same tensorization settings as the prior work of [51].

We observe that our proposed method (ARD) leads to higher accuracy than the train-and-then-compress approach. Our automatic rank determination achieves parameter reduction with only slight accuracy reduction. The CP and TTM methods outperform Tucker and TT methods for this task in terms of accuracy. Previous studies [48, 51] have shown that the compression ratio on convolution layers are often much lower than on fully connected layers due to the small size of convolution filters. Nevertheless, our tensorized training with automatic rank determination always achieves the best compression performance.

4.3.6 Impact: On-Device Training and FPGA Acceleration

Our method can successfully train large end-to-end tensor compressed neural networks and increase the compression ratio during training. End-to-end compressed training has a major impact on edge device training by reducing off-chip memory reads which are an energy and latency bottleneck [17]. In [18] a preliminary FPGA acceleration of our method demonstrates 123× gains in energy efficiency and 59× speedup on a simple two-layer neural network over non-tensorized training on embedded device CPU. These latency and efficiency gains show how our method enables practical on-device training of compact neural networks from scratch. We envision that the performance improvement of on-device training will be more significant on large-scale neural networks. This method may also be implemented with distributed training or on multiple FPGAs to improve the energy efficiency of training huge models on HPC or data centers.

Table 4.8: Training results on the CNN model.

Tensor Type	Model	Training Parameter #	Final Parameter #	Accuracy
	Baseline	13,942,602	13,942,602	90.36
CP	FR	652,748 (21.4×)	652,748 (21.4×)	90.13
	TC-MR [47]	13,942,602 (1×)	9652,748 (21.4×)	75.80
	TC-OR [47]	13,942,602 (1×)	568,412 (24.5×)	71.29
	ARD-LU (Proposed)	1,308,418 (10.6×)	568,412 (24.5×)	90.18
	ARD-HC (Proposed)	1,308,418 (10.6×)	593,419 (23.5×)	90.08
Tucker	FR [52]	653,438 (21.3×)	653,438 (21.3×)	85.15
	TC-MR [48]	13,942,602 (1×)	653,438 (21.3×)	85.36
	TC-OR [48]	13,942,602 (1×)	606,201 (23.0×)	84.86
	ARD-LU (Proposed)	1,307,591 (10.7×)	606,201 (23.0×)	85.41
	ARD-HC (Proposed)	1,307,591 (10.7×)	589,092 (23.7×)	85.86
TT	FR [50]	649,328 (21.5×)	649,328 (21.5×)	87.31
	TC-MR	13,942,602 (1×)	649,328 (21.5×)	86.02
	TC-OR	13,942,602 (1×)	376,123 (37.1×)	85.42
	ARD-LU (Proposed)	1,299,106 (10.7×)	376,123 (37.1×)	86.68
	ARD-HC (Proposed)	1,299,106 (10.7×)	521,096 (26.8×)	85.92
TTM	FR [50]	641,898 (21.7×)	641,898 (21.7×)	90.04
	TC-MR	13,942,602 (1×)	641,898 (21.7×)	81.88
	TC-OR	13,942,602 (1×)	598,693 (22.3×)	80.49
	ARD-LU (Proposed)	1,284,586 (10.9×)	598,693 (22.3×)	90.09
	ARD-HC (Proposed)	1,284,586 (10.9×)	579,217 (24.1×)	90.02

Note: the training parameters in ARD-LU and ARD-HC include posterior mean and variance of every tensorized model parameters, so the number of training variables is 2× of that in fixed-rank tensorized training (FR).

4.4 Conclusion and Future Work

This work has proposed a variational Bayesian method for one-shot end-to-end training of tensorized neural networks. Our work has addressed the fundamental challenge of automatic rank determination, which is important for training compact neural network models on resource-constrained hardware platforms. The customized stochastic variational inference method developed in this paper enables us to train tensorized neural networks with billions of uncompressed model parameters. Our experiments have demonstrated that the proposed end-to-end tensorized training can reduce the training variables by several orders of magnitude. Our proposed method has outperformed all existing tensor compression methods on the tested

benchmarks in terms of both compression ratios and predictive accuracy.

This work will enable ultra memory- and energy-efficient training of AI models on resource-constraint computing platforms, as demonstrated by our preliminary on-FPGA tensorized training in [18]. We will further investigate the theoretical and algorithm/hardware co-design issues in this direction, especially for training large-size neural networks on resource-constraint computing platforms.

4.5 Supplementary Material

This supplementary material provides more technical and experimental details to expand on the methods and experiments presented in this chapter.

4.5.1 Rank Parameter Updates

Firstly we explain the analytical update rules of $\overline{\lambda}_k$ presented in Section 4.3 of the body text. We derive the closed-form updates to a single rank parameter $\overline{\lambda}_k$ for the CP model. The results associated with other tensor formats can be obtained similarly. Firstly, we re-arrange the KL divergence to the prior to isolate all terms involving the rank parameter $\overline{\lambda}_k$:

$$\begin{aligned}
&\mathrm{KL}\Big(q\left(\{\mathbf{U}^{(n)}\},\boldsymbol{\lambda}\right)||p(\{\mathbf{U}^{(n)}\},\boldsymbol{\lambda})\Big)\\
=&\mathrm{KL}\left(q\left(\{\mathbf{U}^{(n)}\}\right)||p\left(\{\mathbf{U}^{(n)}\}\mid\boldsymbol{\lambda}\right)\right)+\mathrm{KL}\left(q\left(\boldsymbol{\lambda}\right)||p\left(\boldsymbol{\lambda}\right)\right)\\
=&\sum_{1\leq n\leq d}\sum_{1\leq i\leq I_n,1\leq j\leq R}\mathrm{KL}\left(\mathcal{N}\left(\overline{u_{ij}^{(n)}},{\Sigma_{ij}^{(n)}}^2\right)||\mathcal{N}\left(0,\lambda_j\right)\right)+\sum_{1\leq j\leq R}\mathrm{KL}\left(\delta(\lambda_j)||p(\lambda_j)\right)\\
\propto&\sum_{1\leq n\leq d}\sum_{1\leq i\leq I_n}\mathrm{KL}\left(\mathcal{N}\left(\overline{u_{ik}^{(n)}},{\Sigma_{ik}^{(n)}}^2\right)||\mathcal{N}\left(0,\lambda_k\right)\right)-p(\lambda_k)\\
\propto&\sum_{1\leq n\leq d}\sum_{1\leq i\leq I_n}\left(\log\sqrt{\lambda_k}+\frac{\overline{u_{ik}^{(n)}}^2+{\Sigma_{ik}^{(n)}}^2}{2\lambda_k}\right)-p(\lambda_k).
\end{aligned}\tag{4.31}$$

Next, we consider a log-uniform rank prior $p(\overline{\lambda}_k)$ and take the derivative of the KL divergence with respect to $\overline{\lambda}_k$. This yields

$$\frac{\partial}{\partial\overline{\lambda}_k}\mathrm{KL}\left(q\left(\{\mathbf{U}^{(n)}\},\boldsymbol{\lambda}\right)||p(\{\mathbf{U}^{(n)}\},\boldsymbol{\lambda})\right)\propto\sum_{1\leq n\leq d}\sum_{1\leq i\leq I_n}\left(\frac{1}{2\overline{\lambda}_k}-\frac{\overline{u_{ik}^{(n)}}^2+{\Sigma_{ik}^{(n)}}^2}{2\overline{\lambda}_k^2}\right)+\frac{1}{2\overline{\lambda}_k}.\tag{4.32}$$

Finally, enforcing the gradient (4.32) to be zero yields a closed-form update:

$$\overline{\lambda}_k^{\star}\leftarrow\frac{M}{D+1}.\tag{4.33}$$

Here we have used the notations

$$D=\sum_n I_n,\quad M=\sum_{1\leq n\leq d}\sum_{1\leq i\leq I_n}\overline{u_{ik}^{(n)}}^2+{\Sigma_{ik}^{(n)}}^2.\tag{4.34}$$

The number of entries controlled by λ_j is D, and M is their combined magnitude and variance.

In the case of the Half-Cauchy prior with scale parameter η, the update is

$$\overline{\lambda}_k^\star \leftarrow \frac{M - \eta^2 D + \sqrt{M^2 + (2D+8)\eta^2 M + \eta^4 D^2}}{2D+2}. \tag{4.35}$$

Decreasing the magnitude of the scale parameter η decreases the magnitude of the update of $\overline{\lambda}_k^\star$, thereby increasing rank-sparsity.

4.5.2 Gradient Updates

In Section 4.4 of the body text, we have provided the gradient update rules for CP tensor factors. Here we provide the gradient update for other three tensor formats in our tensorize neural network training.

Tucker: In Tucker format the gradients take a similar form as in the CP format except the rank parameter is dimension-specific.

$$\begin{aligned}
\nabla_{\Sigma_{ij}^{(n)}} \mathcal{L}(q) &= -z \nabla_{u_{ij}} \log\, p(\mathcal{D}|\boldsymbol{\mathcal{G}}, \{\mathbf{U}^{(n)}\}) - \frac{1}{\Sigma_{ij}^{(n)}} + \frac{\Sigma_{ij}^{(n)}}{\lambda_j^{(n)}} \\
\nabla_{\overline{u_{ij}^{(n)}}} \mathcal{L}(q) &= -\nabla_{u_{ij}} \log\, p(\mathcal{D}|\boldsymbol{\mathcal{G}}, \{\mathbf{U}^{(n)}\}) + \frac{\overline{u_{ij}^{(n)}}}{\lambda_j^{(n)}}
\end{aligned} \tag{4.36}$$

The core tensor $\boldsymbol{\mathcal{G}}$ is updated according to

$$\begin{aligned}
&\nabla_{\Sigma_{i_1 \ldots i_d}} \mathcal{L}(q) \\
&\qquad = -z \nabla_{\Sigma_{i_1 \ldots i_d}} \log\, p(\mathcal{D}|\boldsymbol{\mathcal{G}}, \{\mathbf{U}^{(n)}\}) - \frac{1}{\Sigma_{i_1 \ldots i_d}} + \frac{\Sigma_{i_1 \ldots i_d}}{\sigma_0^2} \\
&\nabla_{\overline{g}_{i_1 \ldots i_d}} \mathcal{L}(q) \\
&\qquad = -\nabla_{\overline{g}_{i_1 \ldots i_d}} \log\, p(\mathcal{D}|\boldsymbol{\mathcal{G}}, \{\mathbf{U}^{(n)}\}) + \frac{\overline{g}_{i_1 \ldots i_d}}{\sigma_0^2}.
\end{aligned} \tag{4.37}$$

Tensor Train: In TT-format the parameters are re-indexed to accommodate a third dimension

in addition to the dimension-specific rank parameters referenced above.

$$\begin{aligned}\nabla_{\Sigma^{(n)}_{ijk}}\mathcal{L}\left(q\right) &= -z\nabla_{g_{ijk}}\log\ p(\mathcal{D}|\{\boldsymbol{\mathcal{G}}^{(n)}\}) - \frac{1}{\Sigma^{(n)}_{ijk}} + \frac{\Sigma^{(n)}_{ijk}}{\lambda^{(n)}_{k}} \\ \nabla_{\overline{g^{(n)}_{ijk}}}\mathcal{L}\left(q\right) &= -\nabla_{g_{ijk}}\log\ p(\mathcal{D}|\{\boldsymbol{\mathcal{G}}^{(n)}\}) + \frac{\overline{g^{(n)}_{ijk}}}{\lambda^{(n)}_{k}}\end{aligned} \tag{4.38}$$

Tensor Train Matrix: Finally, for the TTM format we perform one additional re-indexing.

$$\begin{aligned}\nabla_{\Sigma^{(n)}_{ijkl}}\mathcal{L}\left(q\right) &= -z\nabla_{g_{ijkl}}\log\ p(\mathcal{D}|\{\boldsymbol{\mathcal{G}}^{(n)}\}) - \frac{1}{\Sigma^{(n)}_{ijkl}} + \frac{\Sigma^{(n)}_{ijkl}}{\lambda^{(n)}_{l}} \\ \nabla_{\overline{g^{(n)}_{ijkl}}}\mathcal{L}\left(q\right) &= -\nabla_{g_{ijkl}}\log\ p(\mathcal{D}|\{\boldsymbol{\mathcal{G}}^{(n)}\}) + \frac{\overline{g^{(n)}_{ijkl}}}{\lambda^{(n)}_{l}}\end{aligned} \tag{4.39}$$

Chapter 5

Online KSD Thinning for Compressed Bayesian Learning

The work in this chapter was partially inspired by a challenge described in Figure 3.6. The experiment in Figure 3.6 is a parameter sensitivity study addressing the number of particles required for the SVGD inference algorithm (2.15). This chapter addresses the more general challenge of particle selection (and parameter complexity reduction) for Bayesian inference.

A fundamental challenge in Bayesian inference is how to represent a posterior distribution efficiently. Many non-parametric approaches do so by sampling a large number of points using variants of Markov Chain Monte Carlo (MCMC), which are often afflicted by particle starvation, that is, retaining a large number of particles with small weights. We propose an approach based upon embedding posterior distributions in a reproducing kernel Hilbert space, in which distributional goodness of fit can be efficiently computed via the kernelized Stein Discrepancy (KSD). We propose a MCMC variant that retains only those posterior samples which exceed a KSD threshold, which we call KSD Thinning. We establish the convergence and complexity tradeoffs for several settings of KSD Thinning as a function of the KSD threshold parameter, sample size, and other problem parameters. Finally, we provide experimental comparisons against other nonparametric Bayesian methods that generate low-complexity posterior representations, and observe superior consistency/complexity tradeoffs, which is especially salient in the context of training Bayesian deep learners. Our code is available at `github.com/colehawkins/KSD-Thinning`.

5.1 Introduction

Uncertainty quantification is a key component of automated decision-making. Uncertainty estimates permit risk evaluation and deferral to experts in applications such as medical imaging and autonomous driving. Nonparametric Bayesian inference methods such as Markov Chain Monte Carlo (MCMC) are the gold standard in uncertainty estimation problems, but sample complexity is a major bottleneck in their practical application. MCMC methods use the sample path of an ergodic Markov chain whose invariant distribution is the unknown target to produce a series of samples. These samples are then used to construct an estimate of the unknown target. One major limitation of MCMC is that the samples generated by a transition kernel are correlated, which can lead to redundancy in the constructed estimate. In uncertainty quantification each retained sample corresponds to one expensive forward simulation [112, 113, 114] and in machine learning each retained sample requires the storage and inference costs of a expensive model such as a neural network [24, 66]. Therefore balancing *representation quality* and *representational complexity* is an important tradeoff. In standard MCMC, to ensure statistical consistency, the representational complexity approaches infinity. Classically, to deal with the redundancy issue, one may employ post-hoc "thinning," which discards all but a subset of MCMC samples. This task is sometimes called "quantizing" a posterior distribution [115], especially when the work studies Bayesian cubature [116]. Existing approaches generate a large set of samples (usually via MCMC) and then "thin" the sample set. Doing so may require storing a large number of samples before the final post-processing stage [115, 116] and does not allow the sampler to target a compressed representation during the MCMC iterations.

Traditional thinning is done without any goodness-of-fit metric on the samples, which may ignore gradient and distributional information generated by modern MCMC methods such as Langevin Dynamics or Hamiltonian Monte Carlo [117, 118]. Another disadvantage of this approach is that the sampling mechanism cannot dynamically adapt in accordance with which samples may be discarded, which can cause bias in the model to accumulate with time.

To more adeptly discern which samples to retain during thinning, one may compute metrics

between the empirical measure and the unknown target $\mathbb{P}$. However, in a Bayesian inference context, many popular integral probability metrics (IPM) are not computable. To address this issue, Stein's method restricts selection of IPM to ones that employ the score function of the target [119], which, when combined with reproducing kernel Hilbert Space (RKHS) distributional embeddings [120, 121], define statistics to track the discrepancy between distributions in a computationally feasible manner. This is the case with the maximum mean discrepancy [122] or the *kernelized Stein discrepancy* (KSD) [92]. The RKHS embedding imposes smoothness of the empirical measure estimates (by replacing the dirac Delta in (5.1) with a distance-like kernel function).

Various thinning procedures have been developed based upon RKHS embedding: Stein Thinning [115, 116] takes a full chain $\mathbf{S}$ of MCMC samples as input, and iteratively builds a subset $\mathbf{D}$ by greedily maximizing the KSD at each step. Similarly, Stein Point MCMC [91] selects the optimal sample from a batch of m samples during MCMC sampling: at each step it adds the best of m points to a $\mathbf{D}$. Doing so mitigates both the aforementioned redundancy and representational complexity issues; however [91] only append new points to the existing empirical measure estimates, which may still retain too many redundant points.

By contrast, with both constructive and destructive modifications to the dictionary, one may only retain those points which are statistically significant enough to be required for convergence. The goal of this work is to develop such a method by specifying a budget parameter that determines both the transient dictionary complexity and asymptotic bias of the inference by allowing both constructive and destructive point selection during sampling, rather than after the fact. Most similar to this work is [91][Appendix A.6.5], which develops a non-adaptive add/drop criterion, and whose dictionary size grows linearly with the time step. By contrast, this work develops a scheme such that the number of points grows sub-linearly with the sample size.

In other areas of Bayesian inference, post hoc thinning has also been shown to be effective for identifying which samples among a batch are important. When an explicit likelihood model

Method	Online	Informative	Discard Past Samples	Model Order Growth
MCMC Thinning	✓	✗	✗	n
Stein Thinning [115]	✗	✓	✓	NA
SPMCMC [91]	✓	✓	✗	n
Ours	✓	✓	✓	$o\left(\sqrt{n\log(n)}\right)$

Table 5.1: Methods for Generating Compressed Non-Parametric Representations

is available, methods based upon conditional gradient updates have been proposed [123, 124] to find coresets, i.e., a statistically significant subset among a collection of samples. Similarly, in Gaussian Processes [125] and kernel regression [126], one may reduce the complexity of a nonparametric (possibly unnormalized) distributional representation through offline point selection rules such as Nyström sampling [127], greedy forward selection [128, 129], or inducing inputs [130]. In Gaussian processes and kernel regression, fixing the error incurred by memory reduction rather the complexity of the distributional representation, and allowing both constructive/destructive operations in the dictionary selection, yields improved distributional estimates, especially when processing samples online rather than post hoc [131, 132].

We propose a method that generates a compressed representation by **flexible and online** thinning of a stream of MCMC samples. In contrast to existing online approaches that require linear growth in the size of the active set we require only $o\left(\sqrt{n\log(n)}\right)$ growth to ensure convergence through a novel memory-reduction routine we call Kernelized Stein Discrepancy Thinning (KSDT). We compare our approach and several others in Table 5.1. Our method, described at a high level in Figure 5.1, enables sample-efficient Bayesian learning by directly targeting compressed representations during the sampling process. We make the following specific contributions:

C1. We introduce the first online thinning algorithm that can provide informative removal of past MCMC samples during the sampling process. Our algorithm permits a flexible tradeoff between model order growth, thinning budget, and posterior consistency.

C2. We prove in Theorem 5.3.1 that our thinning method can be applied to existing SOTA

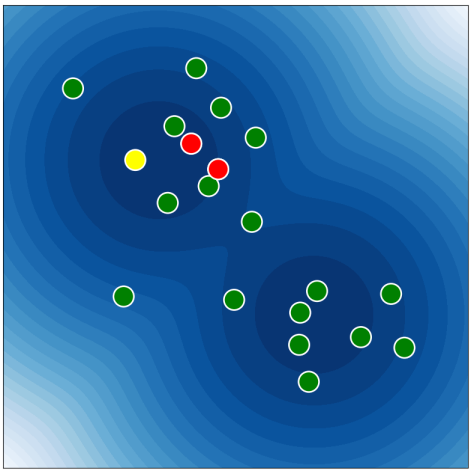


Figure 5.1: Key idea of the the Online KSD Thinning Algorithm. Current dictionary $\mathbf{D}_{t-1}$ in green. New sample $\mathbf{x}_t$ added in yellow to form $\tilde{\mathbf{D}}_t$, rendering red samples redundant. The red samples are pruned.

MCMC algorithms with no change in asymptotic convergence rate when the thinning budget asymptotically decays to 0. In Corollary 5.3.1 we provide the KSD neighborhood of convergence when the thinning budget is fixed.

C3. We test our method on two MCMC problems from the biological sciences and two Bayesian Neural Network problems and demonstrate our thinning algorithm can reduce the number of retained samples but retain or improve baseline sampler performance.

5.2 Online KSD Thinning

5.2.1 Problem Formulation

Given a sequence of points $\mathbf{S} = \{\mathbf{x}_i\}_{i=1}^N$ drawn from an unknown probability measure $\mathbb{P}$ with $\mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^d$, our goal is to infer an approximate empirical measure $q_{\mathbf{D}} = \hat{\mathbb{P}}$. Here $q_{\mathbf{D}}$ is a particle representation with a sparse *dictionary* $\mathbf{D} \subsetneq \mathbf{S}$, i.e., $|\mathbf{D}| \ll N$, where $N \leq \infty$ is the potentially infinite sample size. Specifically, the approximate density associated with $\mathbf{D}$ takes

the form

$$q_{\mathbf{D}}(\cdot) = \frac{1}{|\mathbf{D}|} \sum_{\mathbf{x}_i \in \mathbf{D}} \delta_{\mathbf{x}_i}(\cdot), \tag{5.1}$$

where $\delta_{\mathbf{x}_i}$ denotes the Dirac delta which is 1 if its argument is equal to $\mathbf{x}_i$, and null otherwise.

Our specific focus is on the case that the measure $\mathbb{P}$ admits a density p which can only be evaluated up to an unknown normalization constant. That is, $p = \tilde{p}/Z$ with $\tilde{p}$ as the unnormalized density and $Z > 0$ as the normalization constant. We assume that the unnormalized density $\tilde{p}$ and its score function $\nabla \log \tilde{p}$ may be evaluated in a computationally affordable manner. This set of assumptions is standard in many Bayesian inference problems that arise in machine learning and uncertainty quantification [91, 112, 113, 117].

Our goal is construct $\mathbf{D}$ from the stream of samples $\mathbf{S}$ by determining which points to retain and discard. This empirical distribution may be used to approximate integrals of the form $\int_{\mathcal{X}} f(\mathbf{x})p(\mathbf{x})d\mathbf{x}$ by $\frac{1}{|\mathbf{D}|}\sum_i f(\mathbf{x}_i)$ which arise in computational statistics and uncertainty quantification [24, 112, 113].

5.2.2 Kernelized Stein Discrepancy

As previously mentioned, our focus is on ensuring a constructed empirical measure $q_{\mathbf{D}}$ approximates a target density p associated with samples generated from unknown probability distribution $\mathbb{P}$. In general, without knowledge of the parametric structure of the target distribution, evaluation of metrics between an estimate and target p is intractable. Thus, we focus on the case where the score function of p is available, as with Stein's method [119], and the estimate belongs to a reproducing kernel Hilbert space (RKHS) [120, 121].

The powerful combination of Stein's method and RKHS permits one to evaluate discrepancies between empirical measures in closed form in terms of the kernel associated with an RKHS, as identified in [92]. In particular, denote as k be a base kernel, i.e. radial basis function (RBF) or inverse multi-quadratic (IMQ) which are specified in (5.13) and (5.14). For a fixed target

Algorithm 2 Online Stein Thinning (OST)

Require: Target density p, initial dictionary $\mathbf{D}_0$, point sequence $\mathbf{S} = \{\mathbf{x}_i\}_{i=1}^{\infty}$, budget sequence $\{\epsilon_t\}_{t=1}^{\infty}$, minimum sample size function f.
  for t in $[1, 2, 3, \dots]$ **do**
    Receive new sample $\mathbf{x}_t$ from $\mathbf{S}$
    Add sample $\mathbf{x}_t$ to dictionary: $\tilde{\mathbf{D}}_t = \mathbf{D}_{t-1} \cup \{\mathbf{x}_t\}$
    Prune dictionary via Algorithm 3: $\mathbf{D}_t = KSDT(p, \tilde{\mathbf{D}}_t, \epsilon_t, f(t))$
  end for

density function p, define the positive definite kernel

$$\begin{aligned} k_0(\theta, \theta') =& \nabla_\theta \log p(\theta)^T \nabla_{\theta'} \log p(\theta') k(\theta, \theta') + \nabla_{\theta'} \log p(\theta')^T \nabla_\theta k(\theta, \theta') \\ &+ \nabla_\theta \log p(\theta)^T \nabla_{\theta'} k(\theta, \theta') + \sum_{i=1}^{d} \frac{\partial^2 k(\theta, \theta')}{\partial \theta_i \partial \theta_i'} , \end{aligned} \tag{5.2}$$

where d is the dimension of each particle $\theta, \theta' \in \mathcal{X}$ and the score function $\nabla_\theta \log p(\theta)$ can be estimated without knowledge of the normalizing constants Z. Stein's method in this context specifies a constructed RKHS $\mathcal{K}_0$ with *Stein kernel* k_0 ((5.2)) in turn constructed from the base kernel k. Then the kernelized Stein discrepancy (KSD) of an empirical measure $q_{\mathbf{D}}$ with respect to a target density p is the RKHS norm in $\mathcal{K}_0$ given by

$$\mathrm{KSD}(q_{\mathbf{D}}) = \sqrt{\frac{1}{n^2} \sum_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}} k_0(\mathbf{x}_i, \mathbf{x}_j)}. \tag{5.3}$$

For simplicity our notation suppresses the dependence on the target density p and the RKHS $\mathcal{K}$.

5.2.3 Online Thinning - Outer Loop

We propose an online thinning algorithm that can generate compressed representations of a distribution P with access only to a stream of samples $\mathbf{S}$, the unnormalized density $\tilde{p}$, and the score function $\nabla \log \tilde{p}$. Our sampler performs informative thinning in a flexible online fashion by discarding points that do not make a sufficient contribution to minimizing the KSD objective $\mathrm{KSD}(q_{\mathbf{D}_t})$ where $\mathbf{D}_t$ is the pruned dictionary produced by step t of our algorithm.

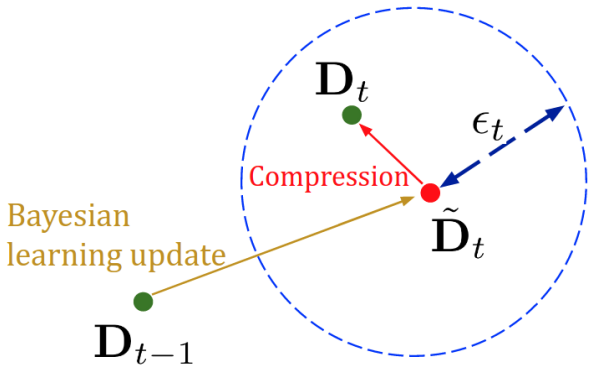


Figure 5.2: A geometric view of our online KSD thinning approach presented in Algorithm 2. We fix an ϵ neighborhood with respect to KSD illuminated in blue, and greedily remove points until we hit the boundary of this neighborhood. By tuning the compression-induced error to the information contained in the update direction, i.e., the yellow arrow representing Equation 5.4, we obtain the red arrow, which is the update output by Algorithm 3

.

Outer Loop: At each time step t we take previous dictionary $\mathbf{D}_{t-1}$ and add a new sample $\mathbf{x}_t$ from our MCMC chain, which results in the expanded auxiliary dictionary:

$$\tilde{\mathbf{D}}_t = \mathbf{D}_{t-1} \cup \{\mathbf{x}_t\} \,, \tag{5.4}$$

which we compress via a destructive inner loop. This outer loop pseudo-code is given in Algorithm 2. If the sample stream $\mathbf{S}$ is generated by an MCMC method and we skip the destructive thinning inner loop then $\mathbf{D}_t = \tilde{\mathbf{D}}_t$ and the update rule exactly matches standard MCMC.

At each step the thinning budget schedule ϵ_t controls the compression/fidelity tradeoff of the pruned approximation. The minimum dictionary size function $f(t)$ must satisfy the relationship $f(t) = o\left(\sqrt{t \log(t)}\right)$ to preserve the consistency our algorithm. However, if one may tolerate a non-vanishing asymptotic bias, then $f(t)$ may be set as a small constant $\epsilon_t = \epsilon$. In practice, one may set $f(t) = \sqrt{t}$ and fix $\epsilon_t = \epsilon$ for some small value of ϵ. We discuss practical selection of ϵ and the resultant dictionary $\mathbf{D}$ in the following section. Our approach differs from the related work of [91] in that we prune the entire dictionary at each step. In contrast, the algorithm presented by [91] receives a batch of samples and selects the best sample from that batch. By

thinning the entire dictionary we can remove previously sampled points that are inessential for ensuring the representation is consistent. A major advantage of this approach is that our online thinning algorithm can automatically determine the number of points required based on the complexity of the target measure $\mathbb{P}$. We demonstrate this in Section 5.4.5.

5.2.4 Online Thinning - Inner Loop

Algorithm 3 Kernelized Stein Discrepancy Thinning (KSDT)

Require: Target distribution p, empirical measure dictionary $\mathbf{D}$, budget ϵ, minimum sample number S
  Compute reference KSD: $M = \mathrm{KSD}(q_{\mathbf{D}})$
  while $\mathrm{KSD}(q_{\mathbf{D}})^2 < M^2 + \epsilon$ and $|\mathbf{D}| > S$ **do**
    Compute least influential point $\mathbf{x}_j$ as in (5.6)
    if $\mathrm{KSD}(q_{\mathbf{D}\setminus\{\mathbf{x}_j\}})^2 < M^2 + \epsilon$ **then**
      Remove least influential point by assigning $\mathbf{D} = \mathbf{D} \setminus \{\mathbf{x}_j\}$
    else
      break loop
    end if
  end while
  return updated dictionary $\mathbf{D}$ satisfying $\mathrm{KSD}(q_{\mathbf{D}})^2 < M^2 + \epsilon$

The inner loop performs destructive thinning on the dictionary $\tilde{\mathbf{D}}_t$ based on a maximum KSD thinning budget ϵ_t and a minimum dictionary size $f(t)$.

Given an intermediate dictionary $\tilde{\mathbf{D}}_t$ our goal is to return a compressed representation $\mathbf{D}_t \subset \tilde{\mathbf{D}}_t$ that satisfies

$$\mathrm{KSD}(q_{\mathbf{D}_t})^2 < \mathrm{KSD}(q_{\tilde{\mathbf{D}}_t})^2 + \epsilon. \tag{5.5}$$

The compressed dictionary $\mathbf{D}_t$ must be ϵ-close in squared KSD to the uncompressed intermediate dictionary $\mathbf{D}_t$. In this section we determine how to build the compressed dictionary $\mathbf{D}_t$.

Related work by [115,116] present *constructive* greedy approaches to subset selection during post-hoc thinning. The challenge with this approach is time complexity, as the inner loop requires $|\tilde{\mathbf{D}}_t|$ point selections in the worst case (no points are pruned). We expect that few points will be pruned and so we follow a destructive approach that requires only $|\mathbf{D}_t| - |\tilde{\mathbf{D}}_t|$

point selections per step. In practice, only 0-2 points are pruned at each step and so $|\mathbf{D}_t| - |\tilde{\mathbf{D}}_t| \in \{0, 1, 2\}$ while $|\mathbf{D}_t|$ ranges from 10-1000.

Our destructive approach iteratively removes points by selecting the "least influential" point in $\mathbf{D}_t$

$$\mathbf{x}_j = \text{argmin}_{\mathbf{x} \in \mathbf{D}} \text{KSD}(q_{\mathbf{D}_t \setminus \{\mathbf{x}\}}) \tag{5.6}$$

If removing the least influential point would violate the KSD criterion in (5.5) we do not prune the point and we break the thinning loop. If removing the least influential point does not violate the KSD criterion we update $\mathbf{D}_t \leftarrow \mathbf{D}_t \setminus \{\mathbf{x}_j\}$ and repeat the thinning procedure. Pseudocode for the full inner loop is given in Algorithm 3 and a visual depiction is given in Figure 5.1.

Thinning a single point in this manner every k steps was suggested by [91] in Appendix A.6.5 of their work. The main difference between their work and ours is flexibility: they do not use a KSD-aware thinning criterion (our budget parameter ϵ_t) and so "good" points may be unnecessarily removed or too few "bad" points will be removed. Their work does not present convergence guarantees for this thinning strategy.

Computational Complexity: The evaluation of (5.6) does not require knowledge of the full symmetric kernel matrix $\mathbf{K}$ whose entries are $\mathbf{K}_{i,j} = k_0(\mathbf{z}_i, \mathbf{z_j})$ for points $\mathbf{z}_i \in \mathbf{D}_{t-1}$. Assuming we maintain two vectors of size $|\mathbf{D}_t|$ (row sums and per-sample KSD contributions) we require $\mathcal{O}(|\mathbf{D}_{t-1}|)$ operations to evaluate (5.6). Evaluating the initial KSD to compute the thinning threshold M requires summing the previous row sums with $\mathcal{O}(|\mathbf{D}_{t-1}|)$ operations. Computing a new row requires $|\mathbf{D}_{t-1}|$ evaluations of k_0. Updating previous row sums and KSD per-sample contributions requires $|\mathbf{D}_{t-1}|$ additions. Thinning requires searching a vector of size $|\mathbf{D}_{t-1} + 1|$ for the maximum KSD contribution. A brute force search requires $\mathcal{O}(|\mathbf{D}_{t-1}|)$ operations. While thinning may occur more than once, we can amortize the cost of thinning across all time steps (since each sample can only be pruned once) to arrive at a worst case per-step complexity of $\mathcal{O}(t)$. In general, the thinning costs are orders of magnitude less expensive than the MCMC sampler costs.

Method	Per-step bound	Per-step dictionary size	Thinning
SPMCMC [91]	$\log(n)/n$	n	✗
Ours	$n\log(n)/f(n)^2$	$f(n) = o\left(\sqrt{n\log(n)}\right)$	✓

Table 5.2: Quantitive Comparison with SPMCMC. Our proposed approach offers a complexity/consistency tradeoff based on the model order growth f and enables online thinning. When $f(n) \propto n$ our approach achieves the same convergence rate as SPMCMC but enables thinning past samples.

5.2.5 Sample Stream Filtering Requirement

In addition to our practical goal of informative online thinning, we aim to provide an algorithm that exhibits provable convergence in KSD. Since we will prune samples, potentially incurring a KSD increase ϵ_t at each step, we require that our sample stream $\mathbf{S}$ already converges in KSD. Therefore we use the Stein Point Markov Chain Monte Carlo (SPMCMC) approach of [91] to generate a sample stream $\mathbf{S}$ that provably and rapidly converges in KSD. The SPMCMC approach has been applied to improve MCMC estimation of parameters for IGARCH modeling of financial time series and inverse problems in uncertainty quantification [91].

SPMCMC Update Rule: Let $\mathbf{D}_{t-1}$ be the current dictionary and $\mathbf{y}_{t,0}$ be the current point. To construct $\mathbf{D}_t$ we initialize an MCMC chain at $\mathbf{y}_{t,0}$, generate m MCMC samples $\mathcal{Y}_t = \{\mathbf{y}_{t,l}\}_{l=1}^m$, and append the KSD-optimal point in $\mathcal{Y}_t$ to $\mathbf{D}_{t-1}$:

$$\begin{aligned} \mathbf{x}_t &= \arg\min_{\mathbf{y}\in\mathcal{Y}_t} \mathrm{KSD}(\mathbf{D}_{t-1} \cup \{\mathbf{y}\}). \\ \mathbf{D}_t &= \mathbf{D}_{t-1} \cup \{\mathbf{x}_t\}. \end{aligned} \tag{5.7}$$

In practice we have found that our method is suitable for many standard samplers, not only SPMCMC samplers. We demonstrate this experimentally in Section 5.4.

5.3 Convergence Results

In this section we provide the convergence guarantees for our proposed KSD-Thinning method (Algorithm 2) under several settings. We consider two sample generation mechanisms: i.i.d. sampling and MCMC sampling, and two settings for the sequence of budget parameters $\{\epsilon_t\}$: decaying budget ($\epsilon_t \to 0$) and fixed budget ($\epsilon_t = \epsilon$). The decaying budget setting is most applicable when one desires exact posterior consistency in infinite time, whereas fixed budgets are useful when a specified limiting error tolerance is sufficient. In both the decaying budget and fixed budget settings, we prove that:

- Our KSD-Thinning algorithm maintains the same asymptotic KSD convergence rate of the baseline sampler, but gains the ability to prune past samples.
- Our algorithm permits a complexity/consistency trade-off that enables drastic reduction in the number of particles in exchange for the cost of a slower asymptotic convergence rate.

The convergence guarantees and complexity/consistency trade-offs of the two thinning budget settings are provided in Theorem 5.3.1 and Corollary 5.3.1, respectively. The most relevant comparisons to our work is [91] which achieves an $\mathcal{O}(\log(n)/n)$ rate of KSD convergence. Our main result in Theorem 5.3.1 differs from the results of [91] in two respects. If we maintain the same asymptotic convergence rate, we can also prune previous samples. Second, our analysis enables the user to trade off between sublinear $o\left(\sqrt{n\log(n)}\right)$ growth of $|\mathbf{D}_t|$ instead of the linear growth in $|\mathbf{D}_t|$ required by [91] at the cost of a slower asymptotic convergence rate. The main technical novelty of our proof is the decomposition of our per-step bound into two terms: the sampling error term from [91] and a thinning error term introduced by Algorithm 3. We defer all proofs to Section 5.7.

5.3.1 Preliminaries

Before we present our main result we introduce two new definitions to impose standard requirements from [91] on the Metropolis-Adjusted Langevin Algorithm (MALA) sampler and the distribution p. Informally, we require that that target density does not possess a heavy tail (distantly dissipative) and that accepted proposals tend to have low norm (inwardly convergent). These conditions are satisfied for the MALA sampler targeting many standard smooth densities (e.g. Gaussian Mixtures).

Definition 5.3.1 *A density p with lipschitz score function $\nabla \log \tilde{p}$ is* **distantly dissipative** *[133] if*

$$\liminf_{r\to 0} \inf_{\mathbf{x},\mathbf{y}} \left\{ \langle \nabla \log \tilde{p}(\mathbf{x}) - \nabla \log \tilde{p}(\mathbf{y}), \mathbf{x}-\mathbf{y}\rangle / \|x-y\|_2^2 \mid \|x-y\|_2 = r \right\} > 0. \tag{5.8}$$

Any density that is strongly log-concave outside of a compact set is distantly dissipative, and a common example is a Gaussian mixture [134].

Definition 5.3.2 *Let $q(\mathbf{x},\mathbf{y})$ denote the MALA transition kernel, and $\alpha(\mathbf{x},\mathbf{y})$ denote the probabilty of accepting proposal $\mathbf{y}$ given current state $\mathbf{x}$. Let $\mathcal{A}(\mathbf{x}) = \{\mathbf{y} \in \mathcal{X} | \alpha(\mathbf{x},\mathbf{y}) = 1\}$. Finally set $\mathcal{I}(\mathbf{x}) = \{\mathbf{y} | \|\mathbf{y}\|_2 \leq \|\mathbf{x}\|_2\}$ A chain generated by the Metropolis-Adjusted Langevin algorithm is* **inwardly convergent** *if*

$$\lim_{\|\mathbf{x}\|_2 \to \infty} \int_{(\mathcal{A}(\mathbf{x}) \cup \mathcal{I}(\mathbf{x})) \setminus (\mathcal{A}(\mathbf{x}) \cap \mathcal{I}(\mathbf{x}))} q(\mathbf{x},\mathbf{y}) d\mathbf{y} = 0. \tag{5.9}$$

In the context of a Gaussian distribution, inward convergence states that when the norm of the current state is large accepted MCMC proposals tend towards the mean. This condition is satisfied in practice by balancing the MALA step size with the norm of the score function. For a thorough discussion see [135].

5.3.2 Results

To achieve convergence we require that the model order $|\mathbf{D}_i|$ is lower bounded by a monotone increasing function $f(i) = o\left(\sqrt{i\log(i)}\right)$. This lower bound on model order growth controls the convergence rate and influences the sequence of maximum possible thinning budgets $\{\epsilon_i\}_{i=1}^{\infty}$.

Theorem 5.3.1 (Decaying Thinning Budget) *Assume that the kernel k_0 satisfies $\mathbb{E}_{\mathbf{y}\sim P}\left[e^{\gamma k_0(\mathbf{y},\mathbf{y})}\right] = b < \infty$, dictionary sizes are lower bounded as $|\mathbf{D}_i| \geq Cf(i)$ where $f(i) = o\left(\sqrt{i\log(i)}\right)$ and that the compression budget $\{\epsilon_i\}$ is $\epsilon_i = \log(i)/f(i)^2$. In either of the following two cases*

- **Case 1:** *Candidate samples $\{\mathbf{y}_{i,l}\}_l^{m_i}$ are drawn i.i.d from the target density p.*
- **Case 2:** *Candidate samples $\{\mathbf{y}_{i,l}\}_l^{m_i}$ are generated by MALA and MALA is inwardly convergent for the distantly dissipative target density p.*

the iterate $\mathbf{D}_n = KSDT(\mathbf{D}_{n-1} \cup \{\mathbf{x}_n\}, \epsilon_n)$ of Algorithm 2 satisfies

$$\mathbb{E}\left[KSD(q_{\mathbf{D}_n})^2\right] \leq C\frac{n\log(n)}{f(n)^2}. \tag{5.10}$$

This result illustrates how both the thinning budget ϵ and the convergence rate depend on f, the model order growth. A faster convergence rate requires larger f, and therefore both faster model order growth and a smaller thinning budget. Our result introduces a trade off between consistency (quality of representation) and complexity (number of particles retained). If we require that the dictionary size grows linearly with $f(i) \propto i$ then we recover the same $\mathcal{O}(\log(n)/n)$ asymptotic convergence rate of Theorem 1 of [91] but gain the ability to prune past samples in the dictionary. If we choose f that grows at a slower rate then the sequence ϵ_i can decay at a slower rate and we can employ more aggressive thinning and achieve a lower-complexity representation at the cost of slower convergence. In practice we found that $\epsilon = 0$ is a simple choice of thinning budget with good empirical performance. When $\epsilon = 0$ thinning

does not increase the KSD. Therefore the required decrease in thinning budget ϵ as f grows is not a concern.

An important practical case is when we desire a fixed error representation of the density p. In this case we wish to set a fixed KSD convergence radius and then determine a constant thinning budget and the required number of steps of Algorithm 2 to achieve a representation of p with KSD error Δ.

Corollary 5.3.1 (Constant Thinning Budget) *Fix the desired KSD convergence radius Δ, assume the same conditions as Theorem 5.3.1, and further assume that $f(i) \propto \sqrt{i^{1+\alpha}\log(i)}$. With constant thinning budget $\epsilon = \mathcal{O}\left(\Delta^{1+\frac{1}{\alpha}}\right)$, after $n = \mathcal{O}\left(\frac{1}{\Delta^{\frac{1}{\alpha}}}\right)$ steps the iterate $\mathbf{D}_n$ produced by Algorithm 2 satisfies*

$$\mathbb{E}\left[KSD\left(q_{\mathbf{D}_n}\right)^2\right] \leq C\Delta \tag{5.11}$$

for some generic constant C. Equivalently, if we fix a constant thinning budget ϵ, after $n = \mathcal{O}\left(\epsilon^{-\frac{2}{\alpha+1}}\right)$ steps the KSD satisfies

$$\mathbb{E}\left[KSD\left(q_{\mathbf{D}_n}\right)^2\right] \leq C\epsilon^{\frac{2}{1+\frac{1}{\alpha}}} \tag{5.12}$$

To our knowledge, this is the first result that specifies the number of steps required to reach a fixed-error KSD-neighborhood of the target density. As in Theorem 5.3.1 our result demonstrates a trade-off between the model order growth and convergence rate. In Theorem 5.3.1 we did not impose a parametric form on f. Here fix the specific parametric form $f(i) \propto \sqrt{i^{1+\alpha}\log(i)}$ for clarity of exposition.

Both Theorem 5.3.1 and Corollary 5.3.1 hold in the more general case of a V-uniformly ergodic MCMC sampler, not just MALA, and we give this result in Theorem 5.7.2 of Section 5.7.

5.4 Experiments

In this section we present the results achieved by applying our thinning mechanism from Algorithm 3 to several samplers, with and without the SPMCMC update rule (see (5.7)). For all experiments we test both the Inverse MultiQuadratic (IMQ) kernel

$$k(\mathbf{x}, \mathbf{y}) = \left(1 + \|\mathbf{x} - \mathbf{y}\|_2^2\right)^{-0.5} \tag{5.13}$$

and the Radial Basis Function (RBF) kernel

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|_2^2 / 2h\right) \tag{5.14}$$

as the base kernels for the KSD as defined in (5.3). The IMQ kernel ensures that the KSD controls convergence in measure [133] unlike the Radial Basis Function (RBF) kernel, so we defer experimental results with the RBF kernel in Section 5.7. We follow [136] and when using the RBF kernel we set $h = \dim(\mathbf{x})$ as the dimension of the input vectors.

In all experiments, the goal is to examine both the fidelity of the representation to $\mathbb{P}$ and the consistency/complexity tradeoff. The KSD is not sufficient for this goal because it does not consider the number of samples retained. If two dictionaries with sizes 10 and 1000 achieve a similar KSD, our metric should select the dictionary of size 10 to reduce computational costs and storage requirements. To address this problem we define a new metric.

From our results (Theorem 5.3.1) and previous works [91] we expect the best-case KSD decay rate of $1/\sqrt{n}$ where n is the number of samples. To characterize the tradeoff incurred by more complex representations we report both the KSD and the **Normalized KSD** $\mathrm{KSD}(q_{\mathbf{D}}) * \sqrt{|\mathbf{D}|}$ which penalizes more complex representations that achieve the same KSD. If a sampler achieves the best-case KSD convergence rate of $1/\sqrt{n}$ we expect the normalized KSD to remain constant. Therefore the Normalized KSD avoids favoring larger dictionaries that are generated by longer (unpruned) sampling runs.

In all experiments we consider three variants of each baseline sampler:

- **Baseline**: This indicates the baseline sampler with no thinning.
- **KSDT-LINEAR**: Our proposed method with linear dictionary growth $f(i) = i/2$.
- **KSDT-SQRT**: Our proposed method with sub-linear dictionary growth $f(i) = \sqrt{i \log(i)}$.

For simplicity we use the constant thinning budget $\epsilon = 0$ in all experiments but note that tuning the dictionary growth rate parameter and thinning budget may improve results. The only hyper-parameter optimization we performed was to tune the baseline samplers for the Bayesian neural network problems.

5.4.1 Goodwin Oscillator

The Goodwin Oscillator [137] is a well-studied test problem for Bayesian inference of ODE parameters [91, 115, 138]. The task is to infer a 4-dimensional parameter that governs the oscillatory feedback mechanisms in a genetic regulatory process, specified by a system of coupled ODEs. A full description can be found in Appendix S5.2 of [115]. We use Random Walk Metropolis (RWM) and Metropolis-Adjusted Langevin Algorithm (MALA) sample chains of length 2×10^6 taken from a public data repository[1] which contains the sample chains used by [115]. We provide the necessary scripts to download and load this data with our code. Since the chains are pre-specified and all KSD-aware sample selection methods are deterministic, we do not present error bars for this experiment. In Figure 5.3 we report the results when applying KSD thinning to both the RWM chain and the RWM chain with the SPMCMC update rule applied. We follow the SPMCMC authors' candidate set size of $m = 10$ (see (5.7)) for this problem [91]. Our thinning methods outperform the baseline samplers on both KSD and Normalized KSD metrics. Our method with linear budget (KSDT-LINEAR) outperforms the square-root budget (KSDT-SQRT) on KSD but not normalized KSD due to the slower dictionary growth rate (smaller number of retained samples) of KSDT-SQRT. Results for the MALA and SPMCMC-MALA methods, as well as results for all samplers with the RBF base

[1] https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/MDKNWM

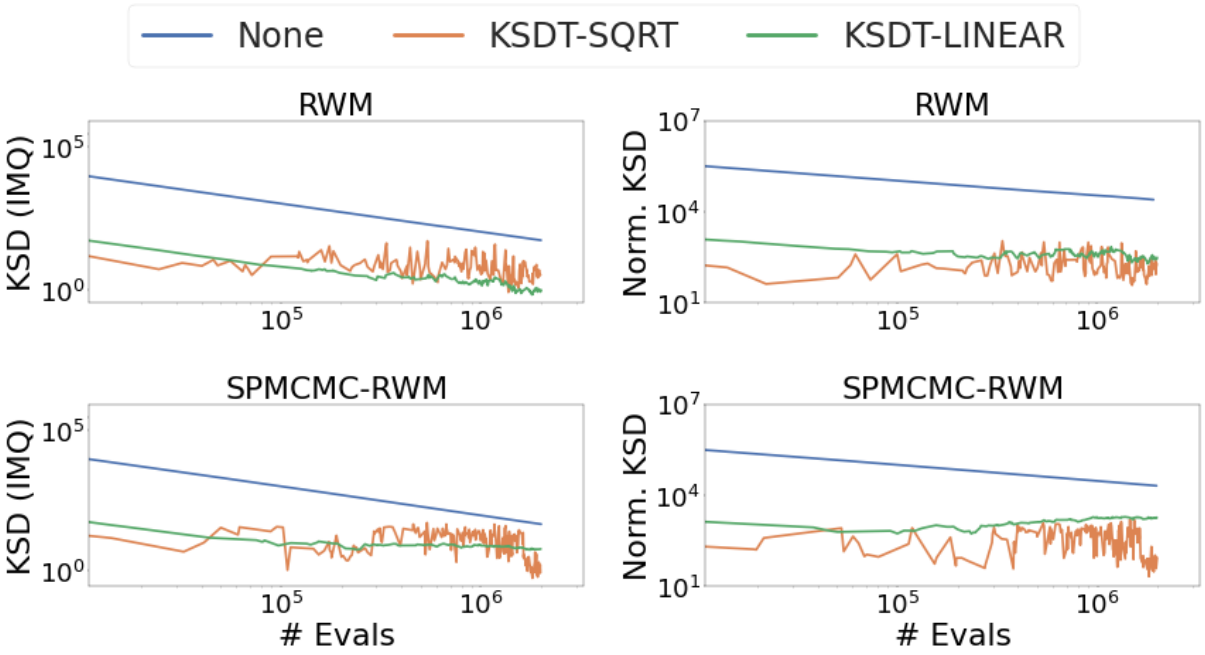


Figure 5.3: Goodwin oscillator problem. Comparison of un-pruned sample chains from the RWM and SPMCMC-RWM samplers with our thinning methods KSDT-SQRT and KSDT-LINEAR. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. Both axes are log-scale. Our KSDT-SQRT/LINEAR methods outperform the baseline methods on both KSD and Normalized KSD.

kernel, are reported in Section 5.5.1 where we detail the effects of online thinning for each individual sampler. In all cases the conclusions are similar to those we draw from Figure 5.3.

5.4.2 Calcium Signalling Model

We consider a calcium signaling model detailed in Appendix S5.4 of [115]. The observed state variables are calcium concentrations and cell membrane potentials, and the task is to infer a 38-dimensional parameter governing the signalling model. Uncertainty in the calcium signalling cascade parameter is used to propagate uncertainty to tissue-level simulations. The dataset consists of 4×10^6 samples of the RWM sampler applied to a tempered posterior distribution. The samples are obtained from the same public repository as the Goodwin model samples[1]. We set the SPMCMC parameter $m = 100$ for this problem since many samples are duplicated due to MCMC rejection. This results in approximately 10 unique samples per step.

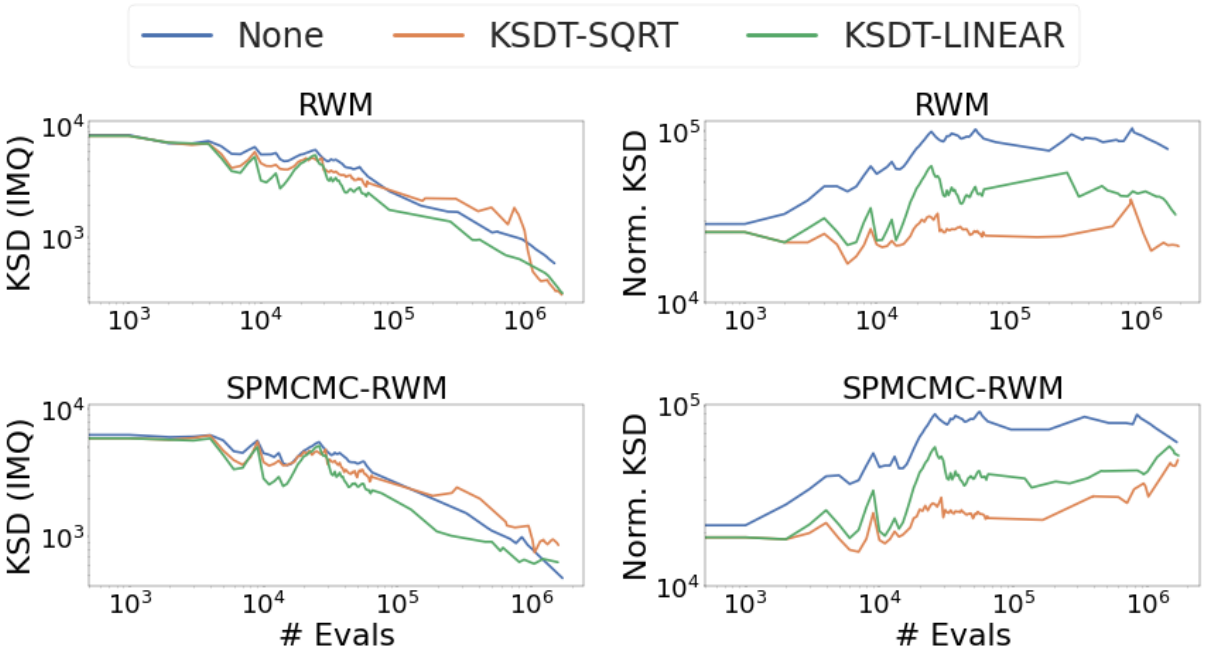


Figure 5.4: Calcium signalling model problem. Comparison of un-pruned sample chains from the tempered RWM and SPMCMC-RWM samplers with our thinning methods KSDT-SQRT and KSDT-LINEAR. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. Both axes are log-scale. Our methods outperform the baseline methods on Normalized KSD, and our method KSDT-LINEAR outperforms the baseline sampler on both KSD and normalized KSD.

We report KSD and Normalized KSD results in Figure 5.4. We observe that KSDT-LINEAR outperforms the baseline sampler on both KSD and normalized KSD in all settings except for the tail end of the SPMCMC-RWM chain. Both KSDT-Linear and KSDT-SQRT outperform the baseline sampler after normalization based on sample complexity is taken into account with the Normalized KSD metric. We repeat this experiment with the RBF base kernel in Figure 5.14 and draw similar conclusions.

5.4.3 Bayesian Neural Network Subspace Inference

Model complexity is a major challenge in sampling-based Bayesian deep learning. Each prediction on unseen test data requires one forward propagation per-sample. Therefore storage and inference costs growly linearly with the number of samples retained. In this section

we demonstrate how our method expands the consistency vs. complexity Pareto frontier for Bayesian deep learning.

MCMC mixing is slow in high dimensions, so a popular technique is to find a low-dimensional subspace and perform Bayesian sampling in that subspace [112, 139]. This idea was applied to Bayesian Neural Network (BNN) training by [140] where the authors construct a two-dimensional "curve subspace" for MCMC sampling. Full details are given in Section 5.5.3. In all experiments our goal is to measure the fidelity of each method's approximation to the predictive distribution corresponding to the true posterior. We generate the ground truth predictive distribution by running 20,000 samples of SGLD [117]. We follow [68] and measure the top-1 agreement and total variation with respect to the ground truth predictive distribution. Top-1 agreement is measured

$$\frac{1}{n}\sum_{i=1}^{n}\max\left(1-\left|\arg\max_{j}p_1(\mathbf{y}_i=j|\mathbf{x}_i)-\arg\max_{j}p_2(\mathbf{y}_i=j|\mathbf{x}_i)\right|,0\right) \tag{5.15}$$

where p_1, p_2 are two predictive distributions and $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ are the test data. Higher is better. Total variation is measured by

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\sum_{j=1}^{c}|p_1(\mathbf{y}_i=j|\mathbf{x}_i)-p_2(\mathbf{y}_i=j|\mathbf{x}_i)| \tag{5.16}$$

where c is the number of classes. Lower is better. Higher agreement and lower total variation indicate higher-fidelity approximations to the ground truth predictive distribution. All results in this section are the mean over 5 chains. We provide additional results on accuracy and calibration in Section 5.5.3, as well as results using the RBF base kernel. No batch normalization or data augmentation is used as they do not admit Bayesian interpretations [141]. In all experiments the hyper-parameters were optimized only for the ground truth MCMC chain. We set the SPMCMC parameter to $m = 5$ and all methods perform 2000 SGLD steps. We measure the agreement, total variation, and the number of dictionary samples at steps $\{100, 200, 500, 1000, 2000\}$ for each task.

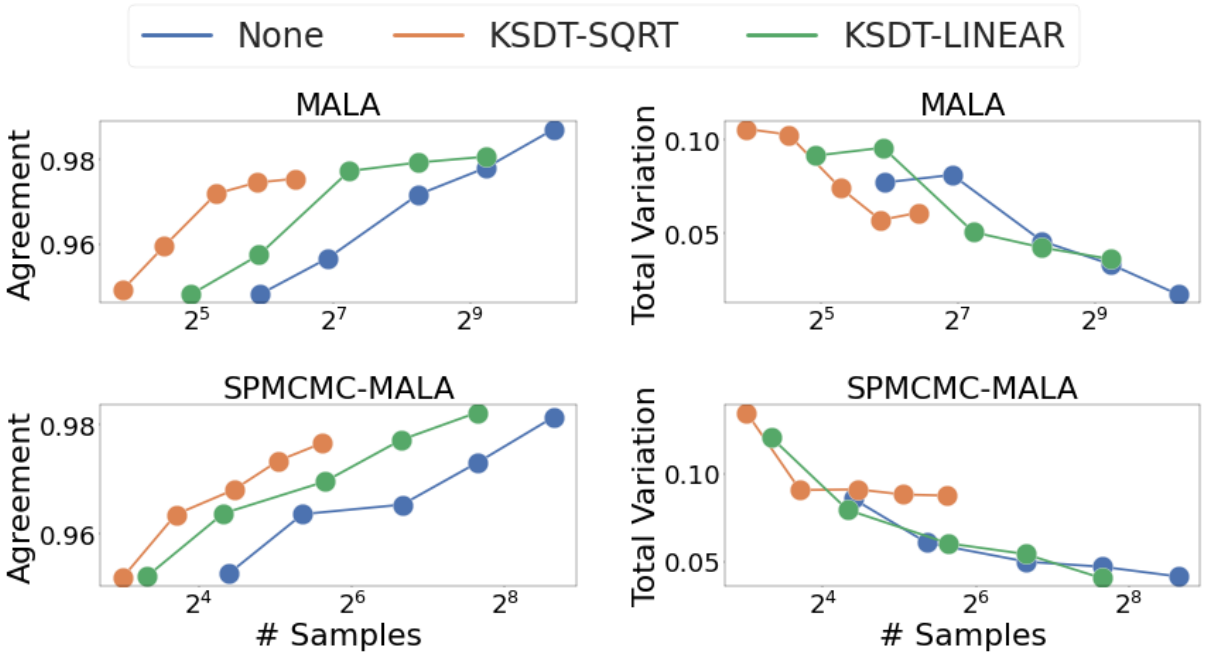


Figure 5.5: Agreement (left) and Total Variation (right) with the ground truth CIFAR-10 posterior predictive distribution. Higher Agreement and lower Total Variation indicate more accurate representations of the posterior predictive distribution. Color indicates thinning method (or no thinning baseline). The x-axis is log-scale. Our KSDT-SQRT/LINEAR methods Pareto-dominate the baseline sampler on the Agreement metric and improve the low-sample Pareto frontier for the Total Variation metric.

CIFAR-10 Classification The task is 10-class image classification, and we use a ResNet-20 with Filter Response Normalization from [68] as our base model. Our results in Figure 5.5 demonstrate that our online thinning method outperforms both the baseline sampler and SPMCMC-based samplers on the agreement metric. Improvement is mixed on the total variation metric, where our methods outperform existing methods in the low-sample regime, but have similar performance to existing methods once the model complexity is high.

IMDB Sentiment Prediction The task is two-class sentiment prediction and we use a CNN-LSTM from [68] as our base model. Our results in Figure 5.6 demonstrate that all of our online thinning methods, in particular KSDT-SQRT, improve the Pareto frontier of the corresponding baseline sampler on both metrics. This conclusion holds across all sample regimes.

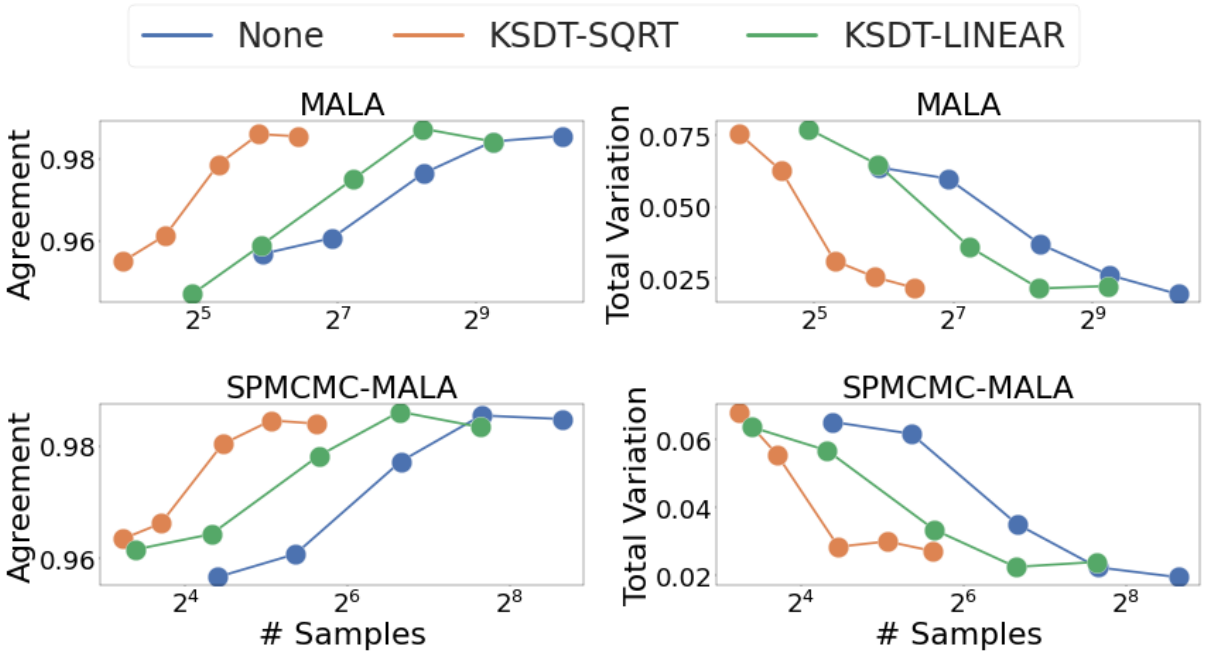


Figure 5.6: Agreement (left) and Total Variation (right) with respect to the ground truth IMDB posterior predictive distribution. Higher Agreement and lower Total Variation indicate more accurate representations of the posterior predictive distribution. Color indicates thinning method (or no thinning baseline). The x-axis is log-scale. Our KSDT-SQRT/LINEAR methods Pareto-dominate the baseline sampler on both the Agreement and Total Variation metrics.

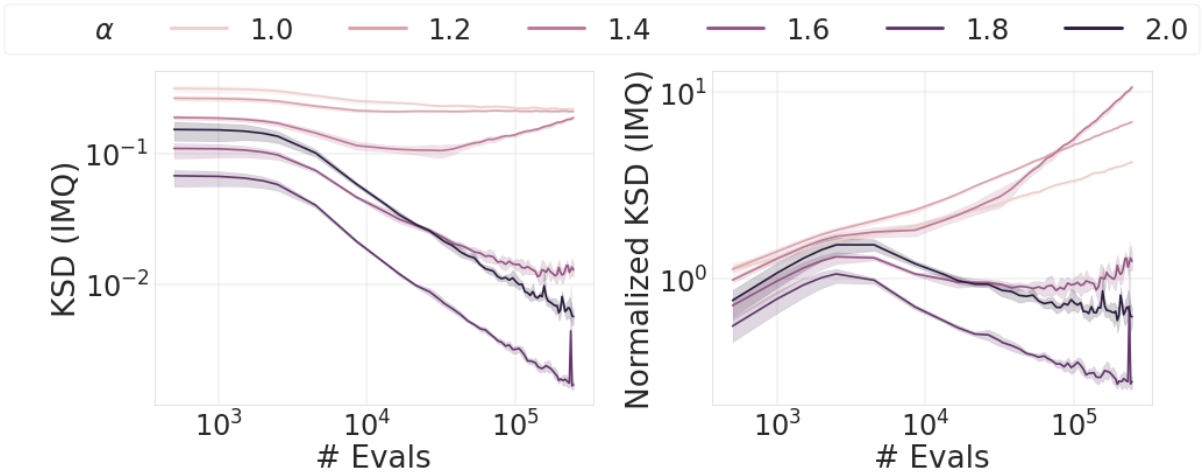


Figure 5.7: Sensitivity to dictionary growth rate parameter α on a bi-modal gaussian mixture problem. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. These results indicate that tuning the dictionary growth rate has potential to further improve our method.

5.4.4 Parameter Sensitivity

We study the sensitivity of our thinning algorithm to the dictionary growth rate $f(i) \in o\left(\sqrt{i \log(i)}\right)$. We target an equally weighted bimodal Gaussian mixture distribution with means $\{(0,0),(1,1)\}$ and covariance $0.5{*}I_2$ for each mode. To decouple the sampler performance from our thinning algorithm we match the i.i.d. sampling setting of Theorem 5.3.1. We draw samples directly from the true distribution and apply the SPMCMC update rule with $m = 5$. In Figure 5.7 we vary the exponent α with budget $f(t) = \sqrt{t^\alpha \log t}/2$. When $\alpha = 2.0$ we use linear growth rate $f(t) = t/2$. We observe that the proposed KSDT algorithm automatically adapts to the target distribution complexity. When the distribution is more complex, KSDT retains for particles. Similar results for the RBF kernel are presented in Figure 5.21.

5.4.5 Automatic Adaptation to Target Complexity

We have studied our KSD-Thinning algorithm with mandatory dictionary growth in order to ensure asymptotic consistency via Theorem 5.3.1. In this section we observe an interesting property of our algorithm: *If we do not enforce model order growth, our thinning algorithm*

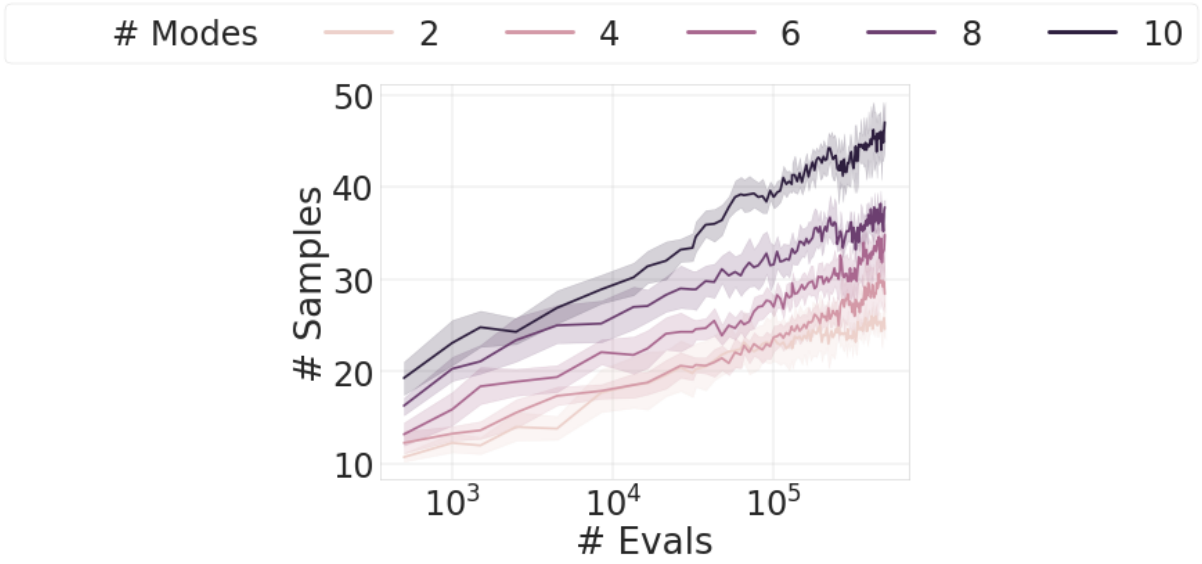


Figure 5.8: Thinning without required dictionary size growth ($f(i) = 10$). As the number of modes of the Gaussian Mixture increases, our algorithm automatically adapts by increasing the number of samples. After 100,000 evaluations our algorithm retains 40 samples for the 10-mode mixture and only 24 for the 4-mode mixture.

can automatically adapt to the target distribution complexity. We set $f(t) = 10$ (the dictionary size cannot decrease below 10) and consider the same setting as Section 5.4.4 but increase the number of modes, with mode i centered at (i, i). In Figure 5.8 we observe that as the number of modes increases, so does the number of retained samples. We replicate this experiment with the RBF kernel in Figure 5.22.

5.5 Additional BNN and RBF Kernel Experiments

In the previous section we focused on results using the IMQ base kernel for the KSD. In this section we report results for the RBF kernel and additional metrics for the Bayesian Neural Network (BNN) problems on both the IMQ and RBF kernels. We keep the same experimental settings as before.

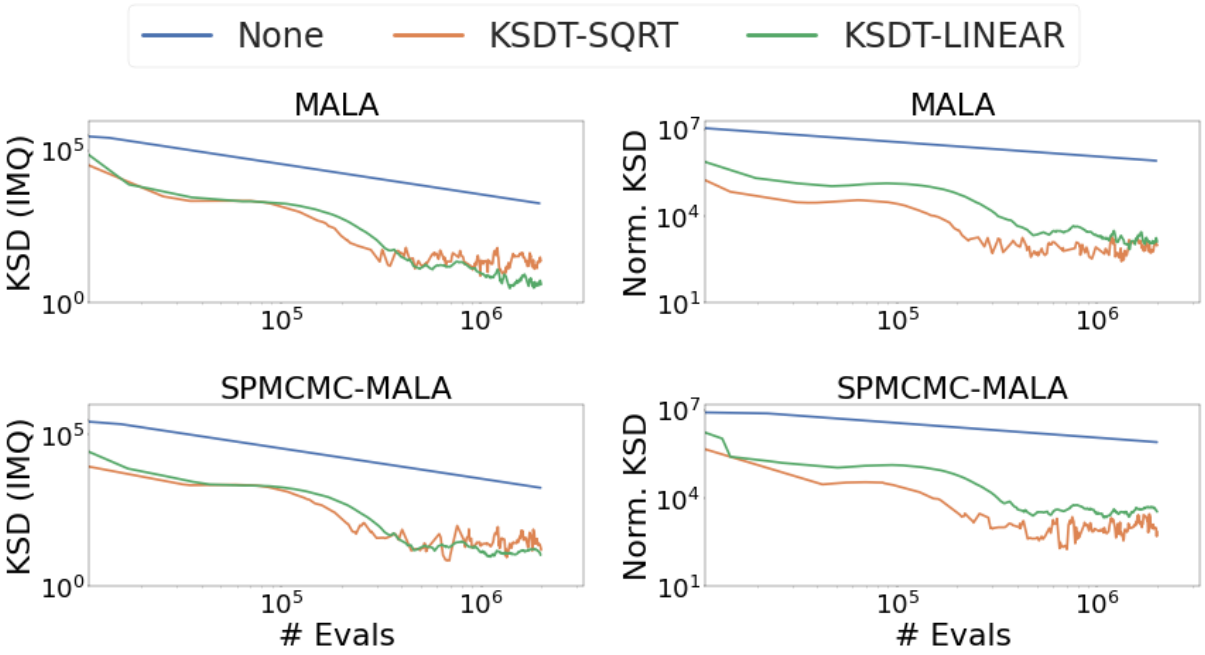


Figure 5.9: Goodwin problem, IMQ base kernel. Comparison of un-pruned sample chains from the MALA and SPMCMC-MALA samplers with our thinning methods KSDT-SQRT and KSDT-LINEAR. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. Both axes are log-scale. Our methods outperform the baseline methods on both KSD and Normalized KSD.

5.5.1 Goodwin Oscillator

We report the results on a per-sampler basis when applying KSD thinning to both the RWM chain and the MALA chains with and without the SPMCMC update rule using either the RBF or IMQ kernels in Figures 5.9,5.11,5.10, and 5.12. In all cases the conclusions are similar to those we draw from Figure 5.3 in the main body. Removing samples using our KSDT-LINEAR and KSDT-SQRT methods improves both KSD and normalized KSD.

5.5.2 Calcium Signalling Model

We report the results on a per-sampler basis when applying KSD thinning to the tempered RWM sampler with and without the SPMCMC update rule using either the RBF or IMQ kernels in Figures 5.13,5.14. In all cases the conclusions are similar to those we draw from Figure 5.4 in the main body. After accounting for model complexity (number of retained

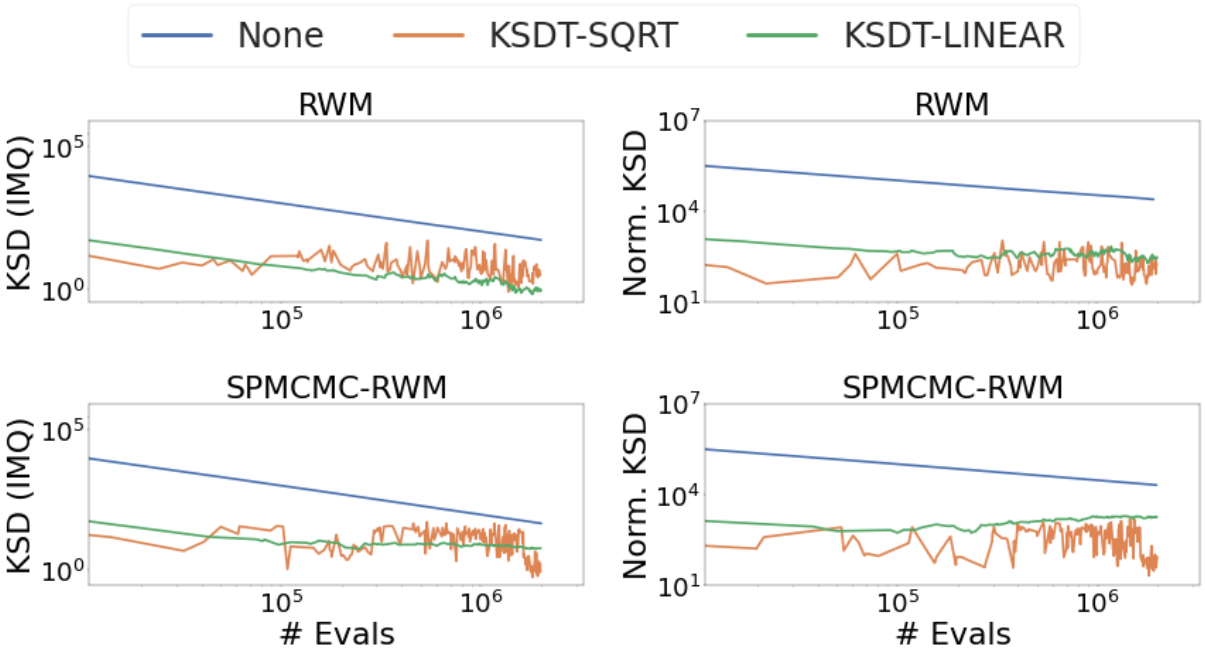


Figure 5.10: Goodwin problem, IMQ base kernel. Comparison of un-pruned sample chains from the RWM and SPMCMC-RWM samplers with our thinning methods KSDT-SQRT and KSDT-LINEAR. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. Both axes are log-scale. Our methods outperform the baseline methods on both KSD and Normalized KSD.

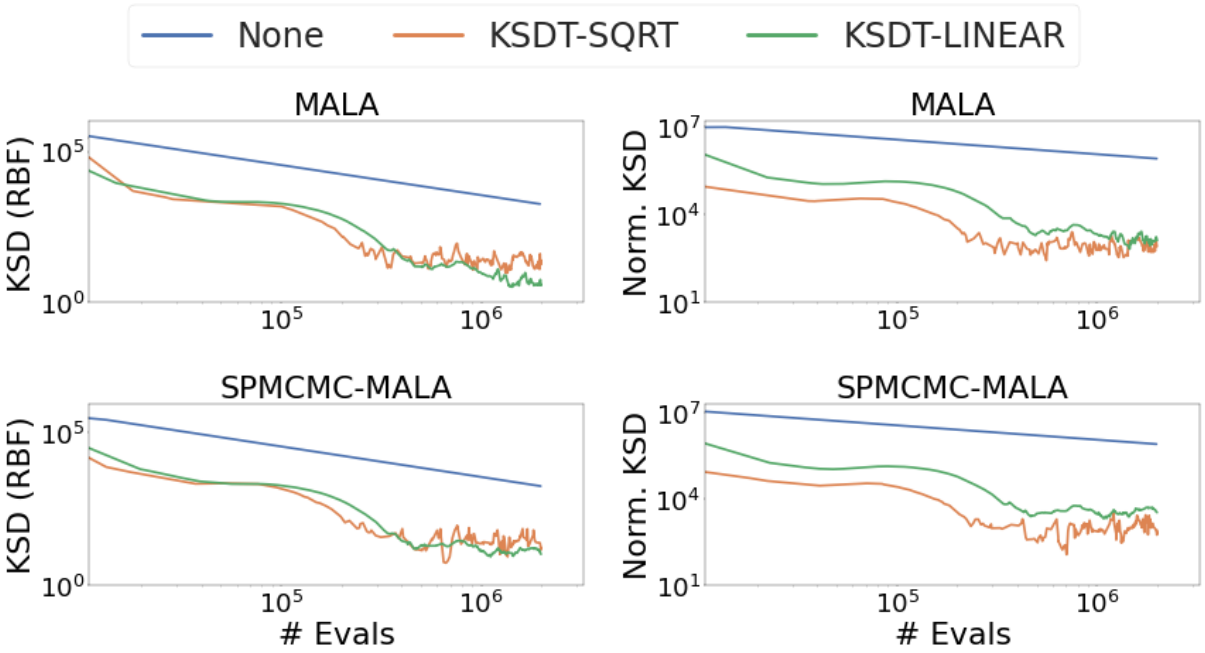


Figure 5.11: Goodwin problem, RBF base kernel. Comparison of un-pruned sample chains from the MALA and SPMCMC-MALA samplers with our thinning methods KSDT-SQRT and KSDT-LINEAR. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. Both axes are log-scale. Our methods outperform the baseline methods on both KSD and Normalized KSD.

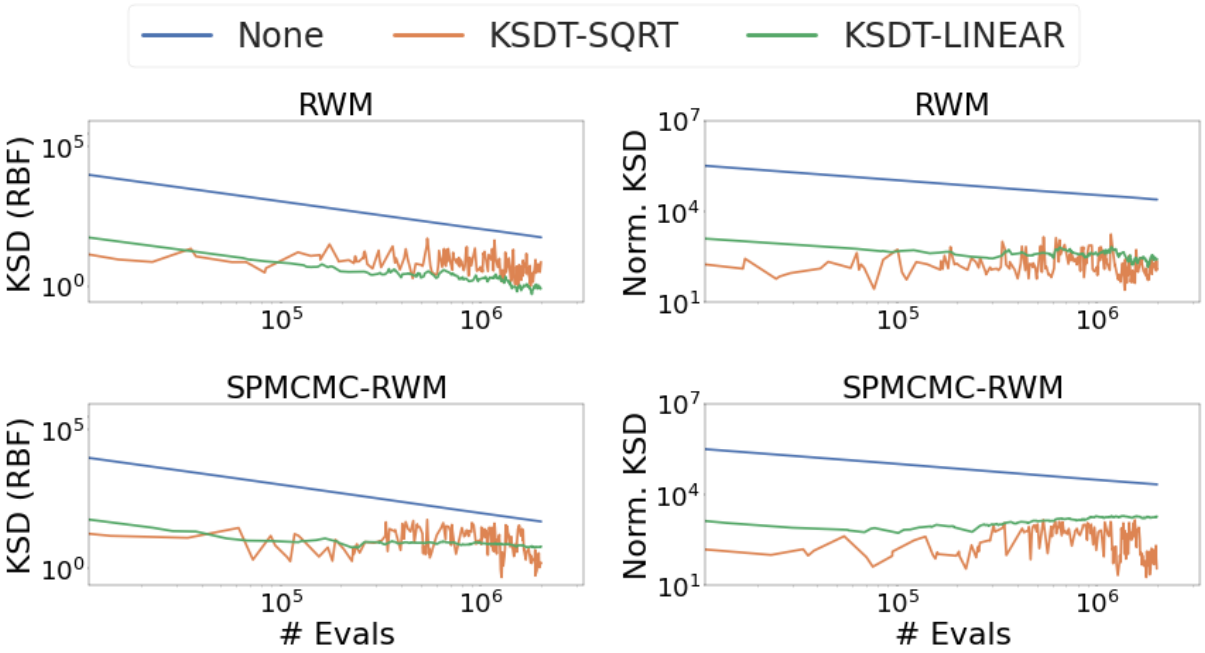


Figure 5.12: Goodwin problem, RBF base kernel. Comparison of un-pruned sample chains from the RWM and SPMCMC-RWM samplers with our thinning methods KSDT-SQRT and KSDT-LINEAR. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. Both axes are log-scale. Our methods outperform the baseline methods on both KSD and Normalized KSD.

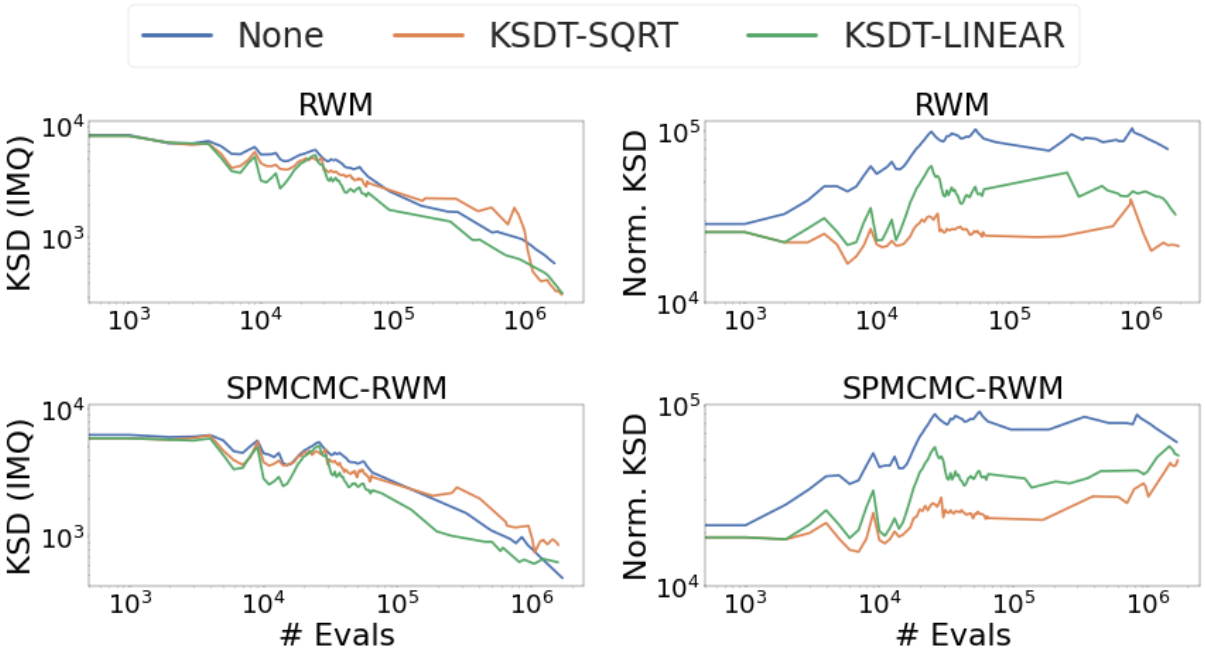


Figure 5.13: Cardiac problem, IMQ base kernel. Comparison of un-pruned sample chains from the tempered RWM and SPMCMC-RWM samplers with our thinning methods KSDT-SQRT and KSDT-LINEAR. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. Both axes are log-scale. Our methods are competitive on KSD and outperform the baseline methods on Normalized KSD.

samples) with the Normalized KSD metric our methods outperform the baseline samplers. Our KSDT-LINEAR and KSDT-SQRT methods outperforms the baseline sampler more often than not on the (un-normalized) KSD metric as well.

5.5.3 Bayesian Neural Network Subspace Sampling

Curve Subspace In order to construct the curve subspace we follow the procedure of [140]. First we pretrain two neural networks with Stochastic Weight Averaging [142]. Next, given the weights $\mathbf{w}_1, \mathbf{w}_2$ of two pretrained networks we initialize the curve midpoint $\mathbf{w}_{1/2} = (\mathbf{w}_1 + \mathbf{w}_2)$ and define the piece-wise linear curve

$$\mathbf{w}_t = \begin{cases} \mathbf{w}_1 + \frac{t}{0.5}\left(\mathbf{w}_{1/2} - \mathbf{w}_1\right) & \text{if } 0 \leq t \leq 0.5 \\ \mathbf{w}_{1/2} + \frac{t-0.5}{0.5}\left(\mathbf{w}_2 - \mathbf{w}_{1/2}\right) & \text{if } 0.5 < t \leq 1.0 \end{cases} \tag{5.17}$$

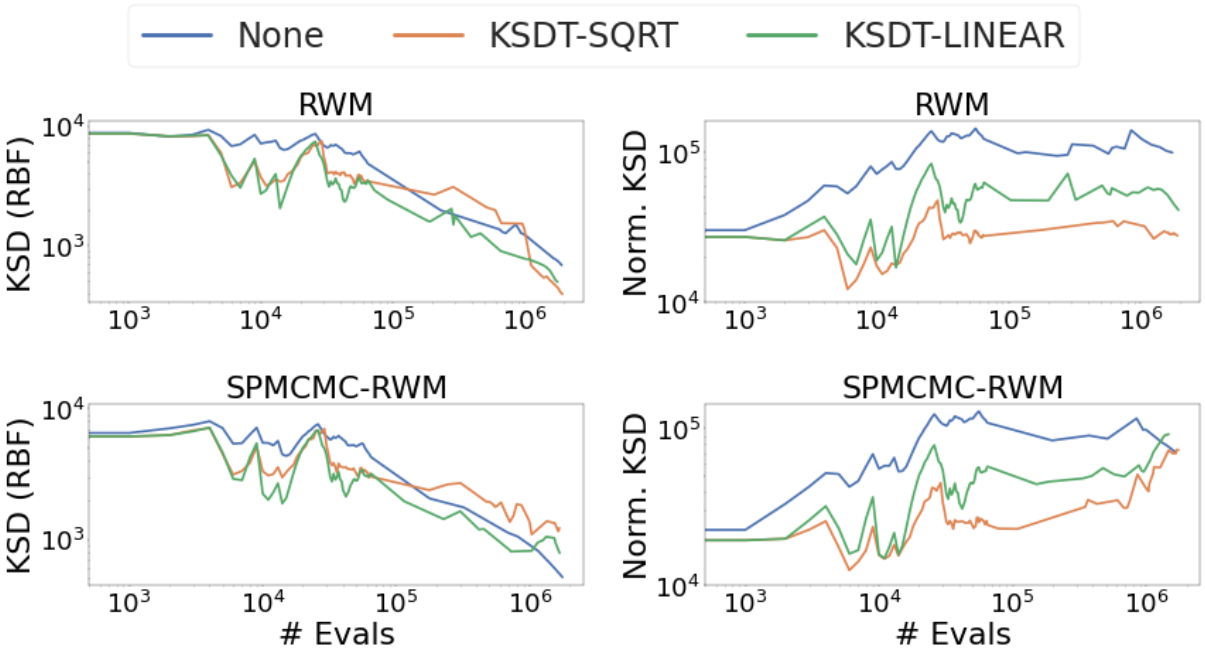


Figure 5.14: Cardiac problem, RBF base kernel. Comparison of un-pruned sample chains from the tempered RWM and SPMCMC-RWM samplers with our thinning methods KSDT-SQRT and KSDT-LINEAR. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. Both axes are log-scale. Our methods are competitive on KSD and outperform the baseline methods on Normalized KSD.

where $t \in (0, 1)$. To train the curve network, for each batch we sample $t \in (0, 1)$ and back-propagate gradients only to $\mathbf{w}_{1/2}$. Finally, we define $\hat{\mathbf{w}} = (\mathbf{w}_0 + \mathbf{w}_1)/2$ as the "base point" and $\mathbf{v}_1 = \mathbf{w}_0 - \hat{\mathbf{w}}$ and $\mathbf{v}_2 = \mathbf{w}_{1/2} - \hat{\mathbf{w}}$ as the subspace vectors. We perform sampling in the 2D subspace centered at $\hat{\mathbf{w}}$ and spanned by the vectors $\mathbf{v}_1, \mathbf{v}_2$.

CIFAR-10 Classification We present results for both total variation and ECE when using the RBF kernel for all KSD-based methods in Figure 5.15 demonstrate that our online thinning method outperforms both the baseline sampler and SPMCMC-based samplers on the agreement metric. We draw the same conclusion as the main body: improvement is mixed on the total variation metric, where our methods outperform existing methods in the low-sample regime, but have similar performance to existing methods once the model complexity is high. In Figures 5.16 and 5.17 we report results for accuracy and expected calibration error (ECE) using the RBF and IMQ kernels respectively. We observe that our thinning methods usually Pareto dominate the baseline methods on accuracy, and offer new points on the ECE Pareto frontier. We note that the goal of all methods is to match the predictive posterior distribution, not necessarily to achieve low accuracy. If the ground-truth predictive posterior has low accuracy and high ECE, then methods that perform best on the task of matching the posterior may have lower accuracy and higher ECE.

IMDB Sentiment Prediction We report results on both total variation and ECE when using the RBF kernel for all KSD-based methods in Figure 5.18. These results imply the same conclusion as the main body text IMQ kernel experiments: our online thinning method outperforms both the baseline sampler and SPMCMC-based samplers on the agreement metric. In Figures 5.19 and 5.20 we report results for accuracy and expected calibration error (ECE) using the RBF and IMQ kernels respectively. We observe that our thinning methods Pareto dominate the baseline methods on accuracy, and offer new points on the ECE Pareto frontier.

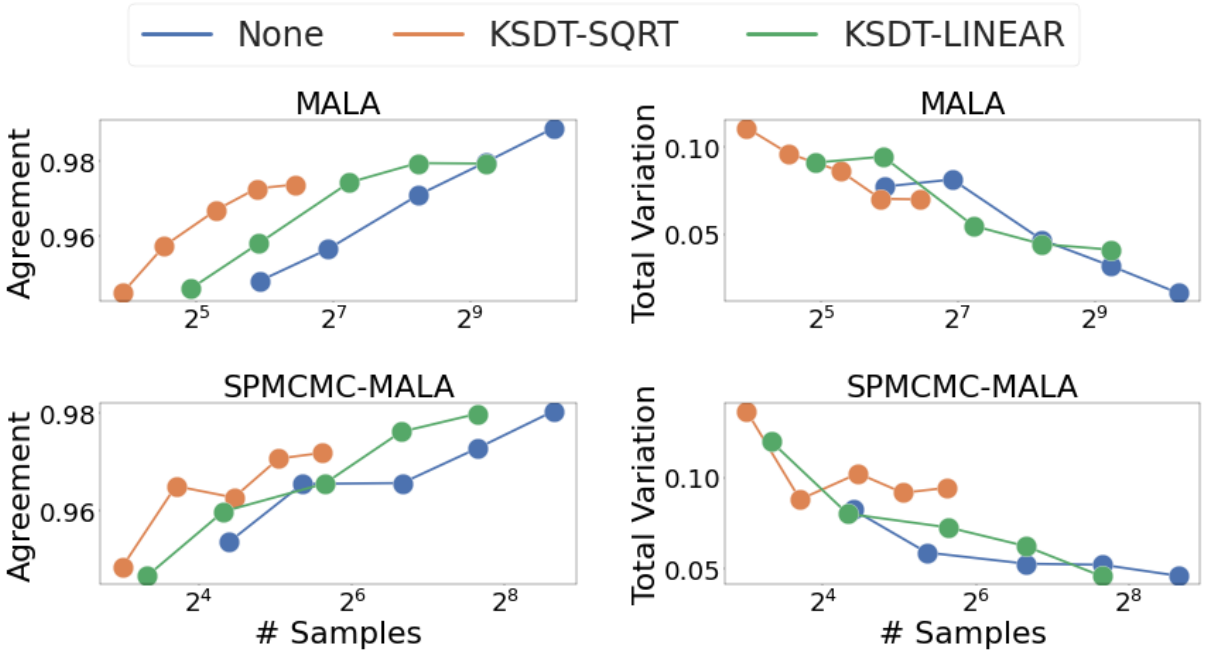


Figure 5.15: Agreement (left) and total variation (right) with the ground truth CIFAR-10 posterior predictive distribution. Higher Agreement and lower Total Variation indicate more accurate representation of the posterior predictive distribution. Color indicates thinning method (or baseline without thinning). The x-axis is log-scale. RBF base kernel. Our methods Pareto-dominate the baseline samplers on the Agreement metric and improve the Total Variation Pareto frontier in the low-sample regime.

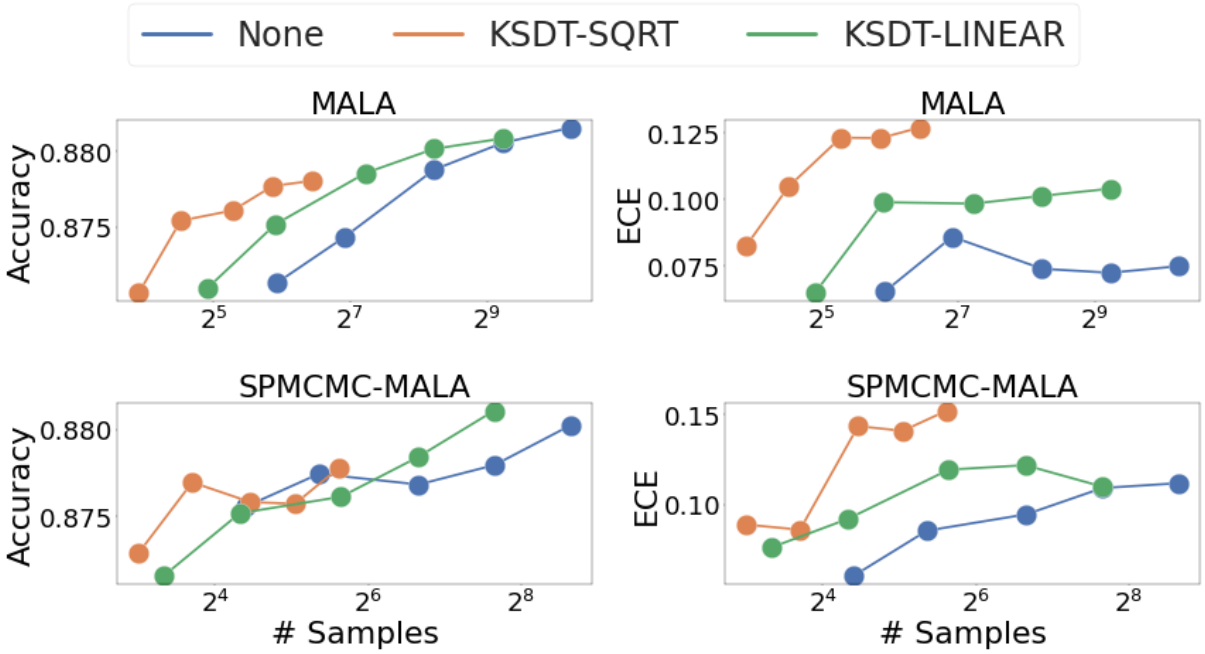


Figure 5.16: Accuracy (left) and Expected Calibration Error (right) on CIFAR-10 using the RBF base kernel. Higher Accuracy and lower ECE are better. Color indicates thinning method (or baseline without thinning). The x-axis is log-scale. Our methods Pareto-dominate the baseline samplers on accuracy and expand the low-sample Pareto frontier for ECE.

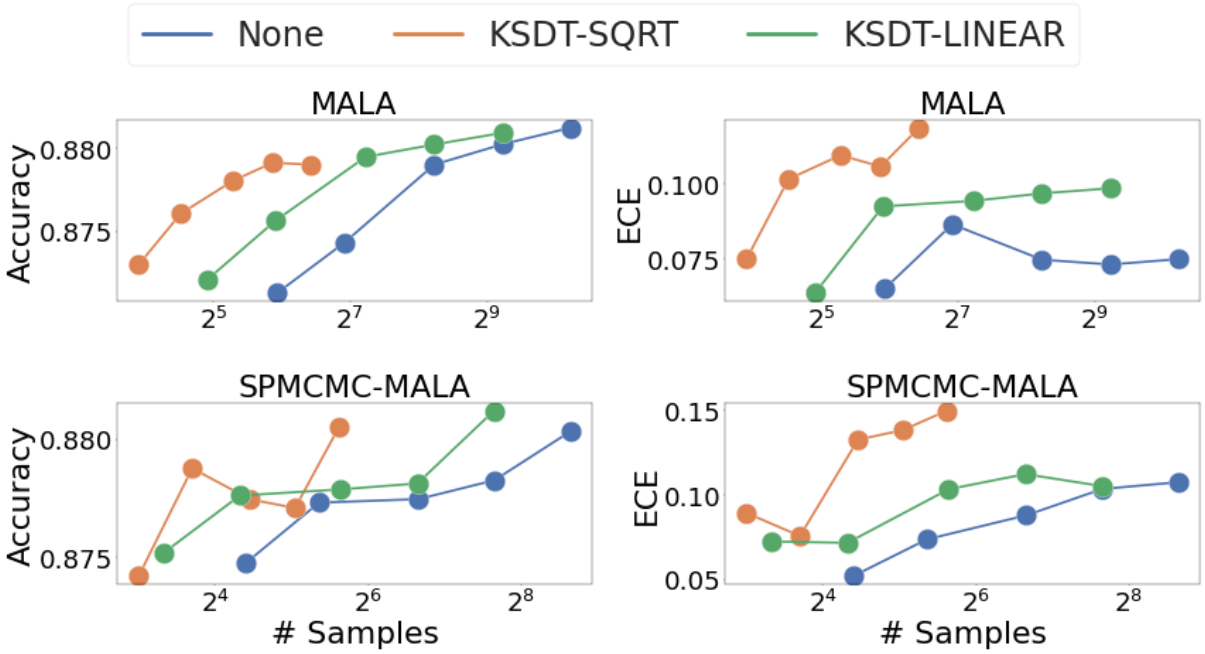


Figure 5.17: Accuracy (left) and Expected Calibration Error (right) on CIFAR-10 using the IMQ base kernel. Higher Accuracy and lower ECE are better. Color indicates thinning method (or baseline without thinning). The x-axis is log-scale. Our methods Pareto-dominate the baseline samplers on accuracy and expand the low-sample Pareto frontier for ECE.

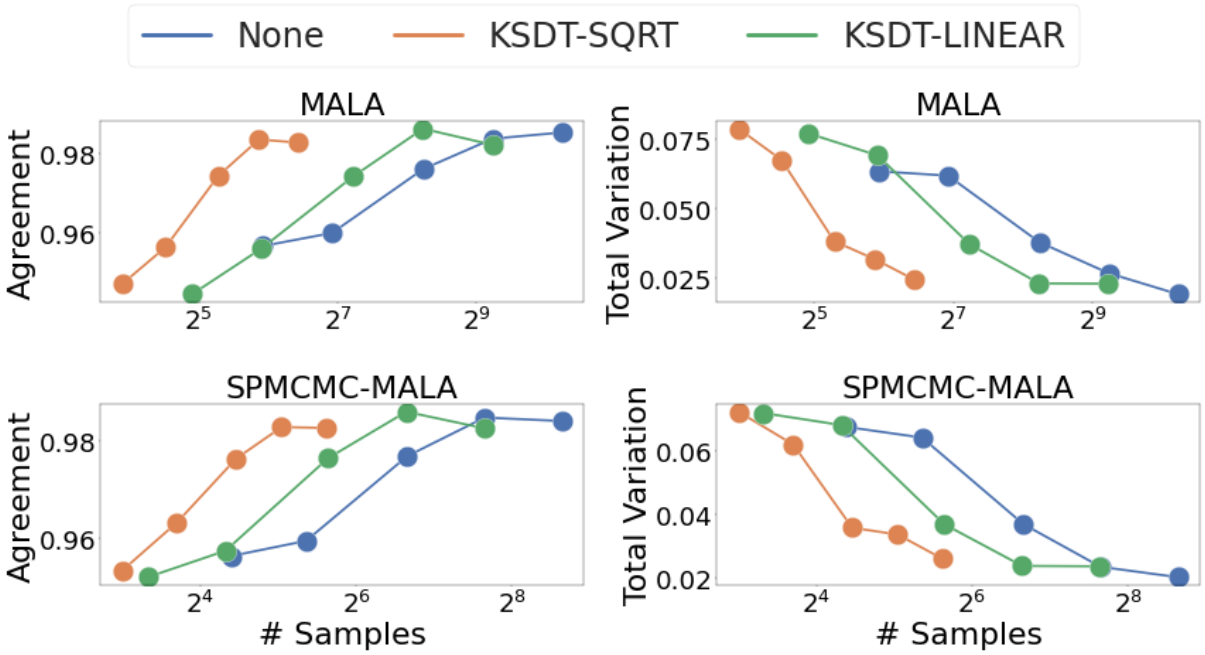


Figure 5.18: Agreement (left) and total variation (right) with the ground truth IMDB posterior predictive distribution. Higher Agreement and lower Total Variation indicate more accurate representation of the posterior predictive distribution. Color indicates thinning method (or baseline without thinning). The x-axis is log-scale. RBF base kernel. Our methods Pareto-dominate the baseline samplers on both the Agreement and Total Variation metrics.

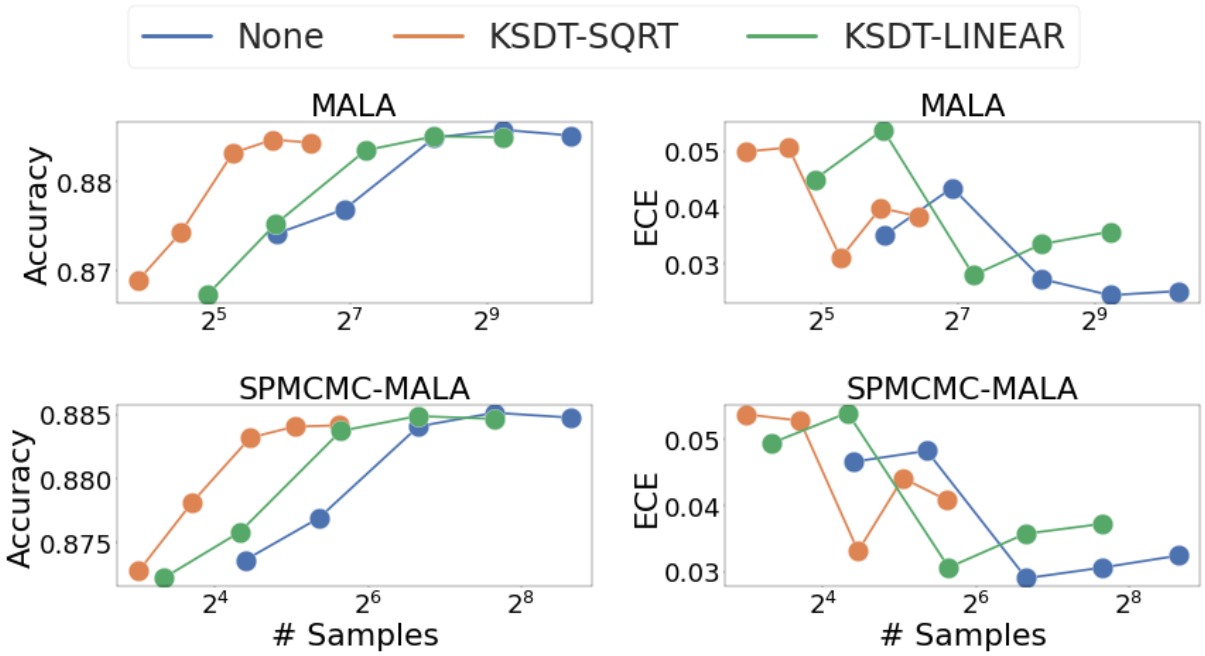


Figure 5.19: Accuracy (left) and Expected Calibration Error (right) on IMDB using the RBF base kernel. Higher Accuracy and lower ECE are better. Color indicates thinning method (or baseline without thinning). The x-axis is log-scale. Our methods Pareto-dominate the baseline samplers on accuracy and expand the low-sample Pareto frontier for ECE.

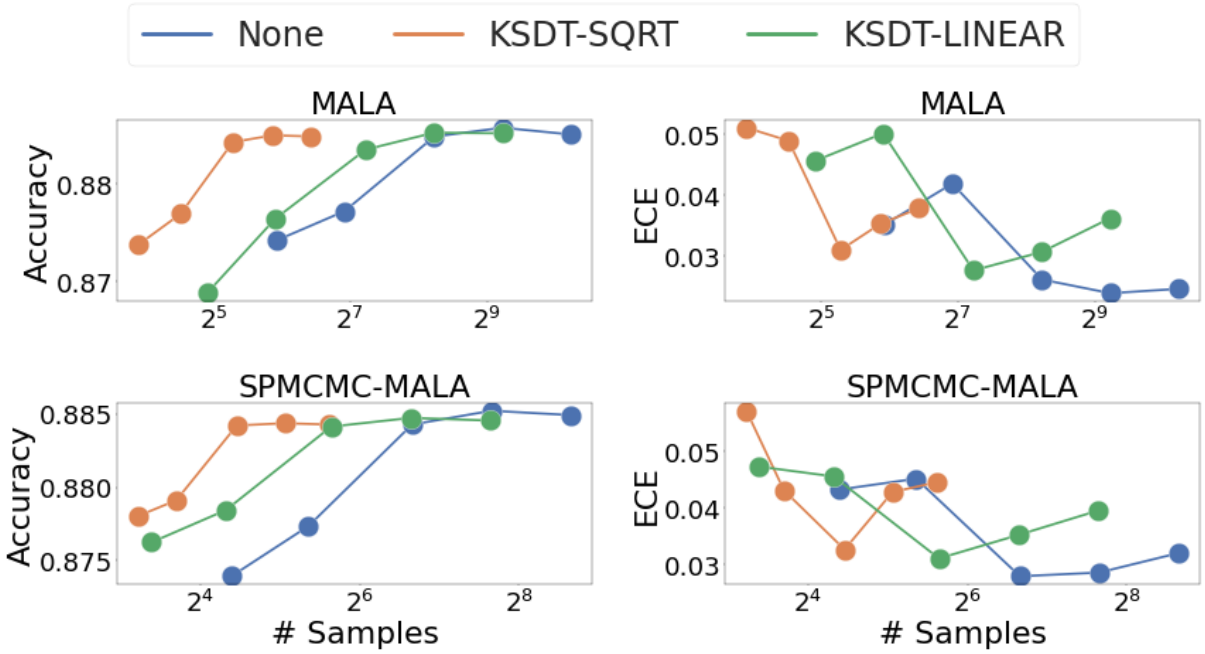


Figure 5.20: Accuracy (left) and Expected Calibration Error (right) on IMDB using the IMQ base kernel. Higher Accuracy and lower ECE are better. Color indicates thinning method (or baseline without thinning). The x-axis is log-scale. Our methods Pareto-dominate the baseline samplers on accuracy and expand the low-sample Pareto frontier for ECE.

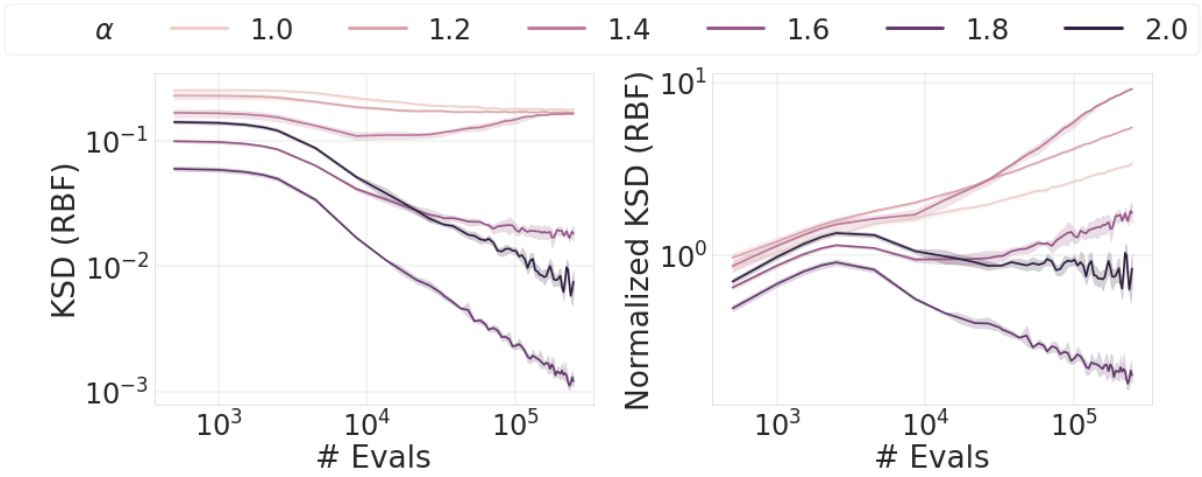


Figure 5.21: Sensitivity to dictionary growth rate parameter α on a bi-modal gaussian mixture problem using an RBF kernel. Lower KSD and lower Normalized KSD indicate more accurate representations of the target distribution. These results indicate that tuning the dictionary growth rate has potential to further improve our method.

5.5.4 Parameter Sensitivity

In Figures 5.21 and 5.22 we repeat the same experiments as Section 5.4.4 with the RBF kernel instead of the IMQ kernel. In Figure 5.7 we vary the exponent α with budget $f(t) = \sqrt{t^{\alpha} \log t}/2$. When $\alpha = 2.0$ we use linear growth rate $f(t) = t/2$. We observe the results are sensitive to parameter tuning on this toy problem, and that similar to the IMQ kernel experiment the best setting for both KSD and normalized KSD is $\alpha = 1.8$.

5.5.5 Automatic Adaptation to Target Complexity

In Figure 5.22 we observe that as the number of modes increases, so does the number of retained samples. This conclusion mirrors the IMQ kernel setting from Section 5.4.5.

5.6 Conclusion

In this paper we presented an online thinning method that produces a compressed representation of a target distribution by thinning "bad" samples during the sampling process. Our work enables MCMC algorithms to directly target complexity-aware representations, and can

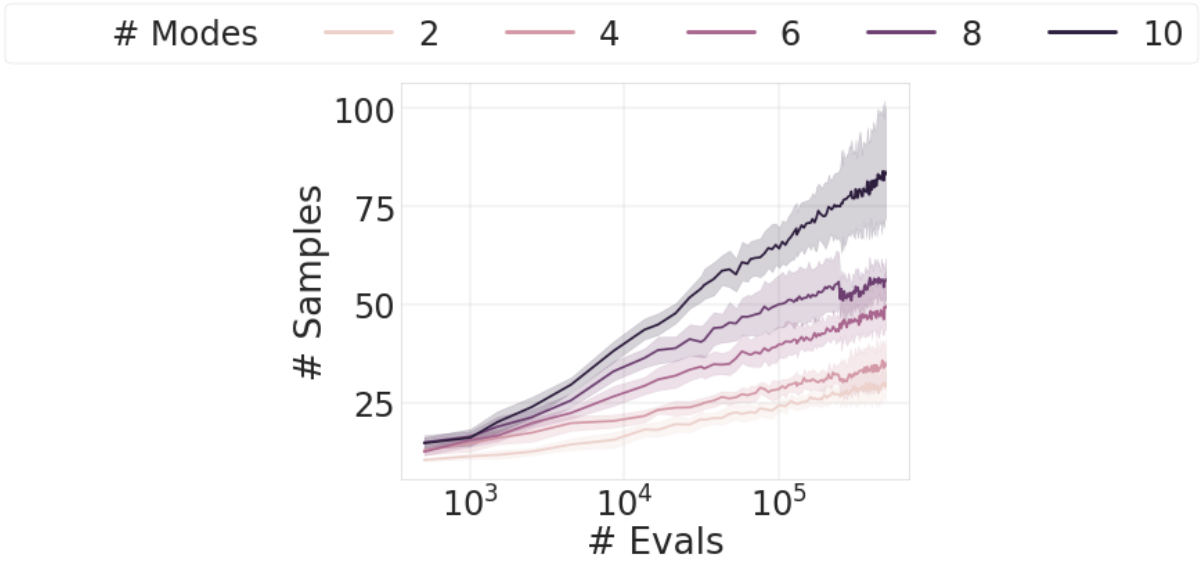


Figure 5.22: Base RBF kernel. As the number of modes of the Gaussian Mixture increases, our algorithm automatically adapts by increasing the number of samples.

be applied to any MCMC sampling scheme when gradients are available. We demonstrated the broad applicability of our method by comparing it to many baseline samplers in several problem settings, and found that it often improves both model fit and reduces model complexity.

Limitations One limitation of our work is that it relies on the KSD, and therefore inherits its limitations. Previous work has shown that in high-dimensions, a careful choice of kernel or lower-dimensional subspace is required for best results [101, 143, 144, 145].

Future directions In this work we introduced our algorithm, but did not explore the tradeoffs of tuning the the thinning budget ϵ in addition to the dictionary growth rate. We focused on settings that are supported by our theoretical results, and believe that tuning the thinning budget and adjusting the dictionary size, independent of theoretical limitations can lead to improved empirical results. Finally, the automatic determination of Bayesian non-parametric model complexity demonstrated in Section 5.4.5 is an interesting direction of future study.

Symbol	Definition
k_0	Stein kernel as defined in (5.2)
$\mathcal{K}_0$	RKHS induced by Stein kernel
KSD	KSD as defined in (5.3)
$\mathcal{Y}_i$	search space for best next point to add
$\mathbf{x}_n$	new point added to dictionary $\mathbf{D}_{t-1}$ before projection step
S_n	"size" of search space Y_i
h_n	Optimal RKHS update given search space $\mathcal{Y}_i$
δ	suboptimality incurred during search for KSD-optimal point
r_n	arbitrary positive constant
m_n	size of new point candidate set
$\{\mathbf{y}_{n,i}\}$	candidate set of MCMC samples
γ, b	constants used to bound exponentiated kernel

Table 5.3: Symbols used for convergence proofs

5.7 Supplementary Material: Proofs of Convergence Results

We provide guarantees of convergence when the sample chain $\mathbf{S}$ is generated by SPCMCMC-style samplers. Our convergence results are provided in expectation, where the expectation $\mathbb{E}$ is taken with respect to the distribution underlying the sample generating process for $\mathbf{S}$. Specifically, the sample stream $\mathbf{S}$ is a sequence of random variables from a time-invariant distribution. Each realization of $\mathbf{S}$ is either a single MCMC sample chain, or the concatenation of several MCMC sample chains according to the SPMCMC update rule given in (5.7).

The two different sampling mechanisms that we consider are (1) i.i.d. sampling from the target distribution and (2) V-uniformly ergodic MCMC sampling. While we do not expect to directly draw samples from the true distribution, the i.i.d. sampling analysis is similar to the MCMC analysis and provides a simpler starting point. The decaying budget result from Theorem 5.3.1 is proven in two parts. Theorem 5.7.1 gives the desired result for i.i.d. sampling and Corollary 5.7.2 gives the desired result for the MALA sampler. A more general result for V-uniformly ergodic MCMC samplers is given in Theorem 5.7.2. The constant-budget result is given for i.i.d. sampling in Corollary 5.7.1 and for the MALA sampler in Corollary 5.7.3.

The following technical lemma imposes constraints on the sample stream $\mathbf{S}$. We will par-

tition $\mathbf{S}$ into a sequence of batches $\mathcal{Y}_i$ and select the KSD-optimal element from each batch. The lemma below is stated for generic constants S_i, r_i and search sets $\mathcal{Y}_i$ to accommodate two cases (1) $\mathbf{S}$ is a stream of i.i.d samples from the target distribution (2) $\mathbf{S}$ is a stream of MCMC samples.

Lemma 5.7.1 (General Recursion Lemma, modified from Theorem 5 in [91]) *Let $\mathcal{X}$ be the domain of p. Fix $n \in \mathbb{N}$. Assume that for all $j \leq n$ the sequence of points $\{\mathbf{x}_i\}_{i=1}^{j}$, where $\mathbf{x}_i$ the greedy KSD-optimal point selected from the search space $\mathcal{Y}_i \subset \mathcal{X}$, satisfies*

$$\frac{k_0(\mathbf{x}_j, \mathbf{x}_j)}{2} + \sum_{\mathbf{x}_i \in \mathbf{D}_{j-1}} k_0(\mathbf{x}_i, \mathbf{x}_j) \leq \frac{\delta}{2} + \frac{S_j^2}{2} + \inf_{\mathbf{x} \in \mathcal{Y}_j} \sum_{\mathbf{x}_i \in \mathbf{D}_{j-1}} k_0(\mathbf{x}_i, \mathbf{x}). \tag{5.18}$$

Then for any constants $\delta > 0$, $\{S_i \geq 0\}_{i=1}^{n}$, $\{r_i > 0\}_{i=1}^{n}$ the pruned dictionary $\mathbf{D}_n = KSDT(\mathbf{D}_{n-1} \cup \{\mathbf{x}_n\}, \epsilon_n)$ satisfies

$$KSD(q_{\mathbf{D}_n})^2 \leq \sum_{i=1}^{n-1} \left(\frac{\delta + S_i^2 + r_i \|h_i\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{i-1}| + 1)^2} + \epsilon_i \right) \left(\prod_{j=i}^{n-1} \frac{|\mathbf{D}_j|}{|\mathbf{D}_j| + 1} \right)^2 \left(\prod_{j=i}^{n-1} \left(1 + \frac{1}{r_{j+1}} \right) \right) \tag{5.19}$$

where h_i is any element of the Stein RKHS corresponding to points in the convex hull of $\mathcal{Y}_i$.

The main distinctions between Lemma 5.7.1 and Theorem 5 of [91] is that we consider additional thinning through Algorithm 3. This thinning operation introduces the budget parameter ϵ_i (potential KSD increase) and the dictionary size $|\mathbf{D}_j|$ since the dictionary size is not a linear monotonically increasing function of the time step as in [91]

Proof The condition in (5.18) assumes that the point sequence $\{\mathbf{x}_j\}$ has bounded suboptimality δ with respect to KSD minimization at each step. Then the conclusion in (5.19) gives an upper bound on the squared KSD at step n. The general constants $\{S_i\}, \{r_i\}$ will be instantiated based on the choice of search spaces $\{\mathcal{Y}_i\}$ and the choice of point generation method (MCMC, deterministic optimization, etc.).

First we will present a bound on the 1-step unpruned iterate. Next we account for KSD

loss due to thinning to achieve a 1-step bound for our algorithm. Finally we recursively apply this bound to achieve the recurrence relation in (5.19).

First we provide a 1-step bound for the unpruned iterate $\tilde{\mathbf{D}}_n$ by adapting the proof style of Theorem 5 in Appendix A.1 of [91] to suit our notation.

$$\begin{aligned}
|\tilde{\mathbf{D}}_n|^2 \mathrm{KSD}(q_{\tilde{\mathbf{D}}_n})^2 &= \sum_{\mathbf{x}_i \in \tilde{\mathbf{D}}_n} \sum_{x_j \in \tilde{\mathbf{D}}_n} k_0(\mathbf{x}_i, \mathbf{x}_j) \\
&= |\mathbf{D}_{n-1}|^2 \mathrm{KSD}(q_{\mathbf{D}_{n-1}})^2 + k_0(\mathbf{x}_n, \mathbf{x}_n) + 2 \sum_{\mathbf{x}_i \in \mathbf{D}_{n-1}} k_0(\mathbf{x}_i, \mathbf{x}_n) \\
&\leq |\mathbf{D}_{n-1}|^2 \mathrm{KSD}(q_{\mathbf{D}_{n-1}})^2 + \delta + S_n^2 + 2 \inf_{\mathbf{x} \in \mathcal{Y}_j} \sum_{\mathbf{x}_i \in \mathbf{D}_{n-1}} k_0(\mathbf{x}_i, \mathbf{x}) \qquad (5.20)
\end{aligned}$$

The equality follows from the definition of the KSD in (5.3). The second equality follows from the definition of the KSD again as we separate the elements in the final row and final column of the kernel matrix to get the second two terms. The last inequality is the direct application of the premise of Lemma 5.7.1, which ensures bounded suboptimality of the point $\mathbf{x}_n$, to the last two terms on the right-hand side of the middle equality in (5.20). Multiply the premise by 2 for direct application.

We apply the following result from Equation (11) of [91]:

$$2 \inf_{x \in \mathcal{Y}_j} \sum_{\mathbf{x}_i \in \mathbf{D}_{n-1}} k_0(\mathbf{x}_i, \mathbf{x}) \leq r_n \|h_n\|_{\mathcal{K}_0}^2 + \frac{\mathrm{KSD}(q_{\mathbf{D}_{n-1}})}{r_n}$$

to the last term on the right-hand side of (5.20) to get

$$|\tilde{\mathbf{D}}_n|^2 \mathrm{KSD}(q_{\tilde{\mathbf{D}}_n})^2 \leq |\mathbf{D}_{n-1}|^2 \left(1 + \frac{1}{r_n}\right) \mathrm{KSD}(q_{\mathbf{D}_{n-1}})^2 + \delta + S_n^2 + r_n \|h_n\|_{\mathcal{K}_0}^2. \qquad (5.21)$$

To recover the un-pruned KSD we divide both sides by both sides by $|\tilde{\mathbf{D}}_n|^2 = (|\mathbf{D}_{n-1}| + 1)^2$ to

get

$$\mathrm{KSD}(q_{\tilde{\mathbf{D}}_n})^2 \leq \frac{|\mathbf{D}_{n-1}|^2}{(|\mathbf{D}_{n-1}|+1)^2}\left(1+\frac{1}{r_n}\right)\mathrm{KSD}(q_{\mathbf{D}_{n-1}})^2 + \frac{\delta + S_n^2 + r_n\|h_n\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{n-1}|+1)^2} \tag{5.22}$$

We replace the un-pruned KSD $\mathrm{KSD}(q_{\tilde{\mathbf{D}}_n})^2$ with the pruned KSD $\mathrm{KSD}(q_{\mathbf{D}_n})^2$ and apply the stopping criterion of Algorithm 3 to establish the recurrence relationship

$$\mathrm{KSD}(q_{\mathbf{D}_n})^2 \leq \frac{|\mathbf{D}_{n-1}|^2}{(|\mathbf{D}_{n-1}|+1)^2}\left(1+\frac{1}{r_n}\right)\mathrm{KSD}(q_{\mathbf{D}_{n-1}})^2 + \frac{\delta + S_n^2 + r_n\|h_n\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{n-1}|+1)^2} + \epsilon_n. \tag{5.23}$$

This recurrence may be unrolled backwards in time to write

$$\mathrm{KSD}(q_{\mathbf{D}_n})^2 \leq \sum_{i=1}^{n}\left(\frac{\delta + S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{i-1}|+1)^2} + \epsilon_i\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\left(\prod_{j=i}^{n-1}\left(1+\frac{1}{r_{j+1}}\right)\right) \tag{5.24}$$

which upper bounds the error incurred by each step of the non-parametric approximation to the target density p, as stated in Lemma 5.7.1. ■

Next we present Lemma 5.7.2 which is a consequence of Lemma 5.7.1 when the KSD-optimal point selection is no longer generic, but instead the specific outcome of the SPMCMC update rule from (5.7) applied to the candidate discrete search space. For the following lemma, we assume that we are given a set of candidate samples from the target density p. We partition those samples into candidate sets $\mathcal{Y}_i = \{\mathbf{y}_{i,l}\}_{l=1}^{m_i}$ of size m_i. Then the sample stream $\mathbf{S} = \{\mathbf{x}_i\}_{i=1}^{\infty}$ is generated by the SPMCMC update from (5.7). The following lemma is a modified form of Theorem 6 from [91].

Lemma 5.7.2 (Pruned Recursion with SPMCMC Update Rule) *Assume the same setup as Lemma 5.7.1. Using the SPMCMC update rule from* (5.7) *the iterate* $\mathbf{D}_n = KSDT(\mathbf{D}_{n-1} \cup \{\mathbf{x}_n\}, \epsilon_n)$ *satisfies*

$$KSD(q_{\mathbf{D}_n})^2 \leq \sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{i-1}|+1)^2} + \epsilon_i\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\left(\prod_{j=i}^{n-1}\left(1+\frac{1}{r_{j+1}}\right)\right) \tag{5.25}$$

Proof In this proof we will apply Lemma 5.7.1 to this specific case by instantiating a specific constant δ related to the search procedure and bounding the search space $\mathcal{Y}_i$ with respect to S_i.

First we truncate and redefine the search set $\mathcal{Y}_n$ by restricting our attention to regions with kernel values bounded by S_n^2: $\mathcal{Y}_n = \{\mathbf{x} \in \mathcal{Y}_n \mid k_0(\mathbf{x}, \mathbf{x}) \leq S_n^2\}$. Then we note that the update rule in Equation 5.7 satisfies the premise ((5.18)) in Lemma 5.7.1 with $\delta = 0$ because the infimum is a search over a finite set of points, and therefore the KSD suboptimality δ of the search procedure is $\delta = 0$.

$$\begin{aligned}\frac{k_0(\mathbf{x}_n, \mathbf{x}_n)}{2} + \sum_{\mathbf{x}_i \in \mathbf{D}_{n-1}} k_0(\mathbf{x}_i, \mathbf{x}_n) &= \inf_{\mathbf{x_n} \in \mathcal{Y}_n} \frac{k_0(\mathbf{x}_n, \mathbf{x}_n)}{2} + \sum_{\mathbf{x}_i \in \mathbf{D}_{n-1}} k_0(\mathbf{x}_i, \mathbf{x}_n) \\ &\leq \frac{S_n^2}{2} + \inf_{\mathbf{x} \in \mathcal{Y}_n} \sum_{\mathbf{x}_i \in \mathbf{D}_{n-1}} k_0(\mathbf{x}_i, \mathbf{x}) \end{aligned} \tag{5.26}$$

The first equality follows from the optimality condition of the update rule (select the best point). The second line follows from the truncation criterion of $\mathcal{Y}_n$. Finally we can apply Lemma 5.7.1 with $\delta = 0$ to achieve the desired conclusion in (5.25). This concludes the proof of Lemma 5.7.2. ■

Lemma 5.7.1 is a stepping stone to Lemma 5.7.2. We use Lemma 5.7.2 to establish convergence of Algorithm 2 in the i.i.d. sampling setting of Theorem 5.7.1 by bounding the summation in (5.25) and therefore ensuring KSD convergence.

To achieve convergence we will require that the model order $|\mathbf{D}_i|$ is lower bounded by a monotone increasing function $f(i) \in o\left(\sqrt{i \log(i)}\right)$. The lower bound on model order growth controls the convergence rate and influences maximum possible thinning budget $\{\epsilon_i\}$. This lower bound contrasts with the standard linear $|\mathbf{D}_i| = i$ growth rate for MCMC or i.i.d. sampling and the linear $f(i) \in \mathcal{O}(i)$ growth rate of [91].

Theorem 5.7.1 (i.i.d Sampling and Decaying Thinning Budget) *Assume the same conditions as Lemma 5.7.2. Further assume that the kernel k_0 satisfies $\mathbb{E}_{\mathbf{y} \sim P}\left[e^{\gamma k_0(\mathbf{y}, \mathbf{y})}\right] = b < \infty$, dictionary sizes are lower bounded as $|\mathbf{D}_i| \geq C f(i)$ where $f(i) = o\left(\sqrt{i \log(i)}\right)$ and that the com-*

pression budget $\{\epsilon_i\}$ *is* $\epsilon_i = \log(i)/f(i)^2$. *Finally assume that the candidate samples* $\{\mathbf{y}_{i,l}\}_l^{m_i}$ *are drawn i.i.d from the target density p. Then the iterate* $\mathbf{D}_n = KSDT(\mathbf{D}_{n-1} \cup \{\mathbf{x}_n\}, \epsilon_n)$ *of Algorithm 2 satisfies*

$$\mathbb{E}\left[KSD(q_{\mathbf{D}_n})^2\right] \leq C\frac{n\log(n)}{f(n)^2}. \tag{5.27}$$

Proof To complete our proof of convergence first we will split the recurrence from Lemma 5.7.1 into two terms, one corresponding to the sampling error and one corresponding to the thinning error. Then we will bound each individually by selecting specific values of $\{r_i\}, \{S_i\}$ and $\{m_i\}$ and applying a bound for $\|h_i\|_{\mathcal{K}_0}$.

First we apply standard log-sum-exp bound as in [91] the part related to the sampling error to write

$$\prod_{j=i}^{n-1}\left(1+\frac{1}{r_{j+1}}\right) \leq \exp\left(\sum_{j=1}^{n}\frac{1}{r_j}\right). \tag{5.28}$$

We substitute (5.28) into the conclusion of Lemma 5.7.2 to get

$$\mathbb{E}\left[\mathrm{KSD}(q_{\mathbf{D}_n})^2\right] \leq \mathbb{E}\left[\exp\left(\sum_{j=1}^{n}\frac{1}{r_j}\right)\sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{i-1}|+1\right)^2}+\epsilon_i\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] \tag{5.29}$$

For now we ignore the leading exponential and decompose the summation into two terms related to the sampling and compression-based error, respectively, as:

$$\begin{aligned}&\sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{i-1}|+1\right)^2}+\epsilon_i\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2 \\ &\quad= \underbrace{\sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{i-1}|+1\right)^2}\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2}_{T_1} + \underbrace{\sum_{i=1}^{n}\epsilon_i\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2}_{T_2}\end{aligned} \tag{5.30}$$

The term T_1 is the bias incurred at each step of the unpruned point selection scheme. The term T_2 is the bias incurred by the thinning operation carried out at each step.

Bound on T_1 The term T_1 is the bias incurred by the point sampling scheme.

$$\begin{aligned}
&\mathbb{E}\left[\sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{i-1}|+1\right)^2}\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] \\
&\qquad = \mathbb{E}\left[\sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{n-1}|+1\right)^2}\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_{j-1}|+1}\right)^2\right] \\
&\qquad \leq \mathbb{E}\left[\sum_{i=1}^{n}\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{n-1}|+1\right)^2}\right] \\
&\qquad C\sum_{i=1}^{n}\frac{S_i^2 + r_i\mathbb{E}\left[\|h_i\|_{\mathcal{K}_0}^2\right]}{f(n)^2}
\end{aligned} \tag{5.31}$$

The first equality is a re-arrangement of the denominators that pulls in the term $\frac{1}{(|\mathbf{D}_{i-1}|+1)^2}$. The second inequality is a consequence of the fact that $|\mathbf{D}_j| \leq |\mathbf{D}_{j-1}|+1$ so each product of the dictionary order ratios is at most 1. The final term is based on the assumption of dictionary model order growth, i.e., $|\mathbf{D}_n| \geq f(n)$. From Appendix A, Equation (17) of [91] we can apply truncated kernel mean embeddings and the fact that the points are drawn i.i.d. from the true distribution P to bound $\mathbb{E}\left[\|h_i\|_{\mathcal{K}_0}^2\right]$ by

$$\mathbb{E}\left[\|h_i\|_{\mathcal{K}_0}^2\right] \leq \frac{4b}{\gamma}e^{-\frac{\gamma}{2}S_i^2} + \frac{4}{m_j}S_i^2 \tag{5.32}$$

which relies on the assumption that k_0 satisfies $\mathbb{E}_{\mathbf{y}\sim P}\left[e^{\gamma k_0(\mathbf{y},\mathbf{y})}\right] = b < \infty$. Therefore

$$\begin{aligned}
&\mathbb{E}\left[\sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{i-1}|+1\right)^2}\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] \\
&\qquad \leq C\frac{1}{f(n)^2}\sum_{i=1}^{n}\left((1+\frac{r_i}{m_i})S_i^2 + r_i e^{-\frac{\gamma}{2}S_i^2}\right)
\end{aligned} \tag{5.33}$$

The inequality is an application of (5.32) to the final conclusion of (5.31) followed by an absorption of the constants into C. Note that $C \propto \exp\sum_{i=1}^{n}\frac{1}{r_i}$. We follow [91] and select and choose $S_i = \sqrt{\frac{2}{\gamma}\log(n \wedge m_i)}$ and $r_i = n$. The selection $r_i = n$ is necessary to ignore the leading exponential $\exp\left(\sum_{j=1}^{n}\frac{1}{r_j}\right)$ and render that term constant. The choices of r_i and S_i are

artefacts of the analysis, but do not affect the practical algorithm. Set $m_i = n$ to get

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{i-1}|+1)^2}\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] \\ &\leq C\frac{1}{f(n)^2}\sum_{i=1}^{n}(\log(n)+1) \\ &\leq C\frac{n\log(n)}{f(n)^2}. \end{aligned} \tag{5.34}$$

The first inequality is the conclusion of the previous equation with S_i and r_i evaluated and all constants absorbed. The second inequality holds because the summand does not depend on the index so we can multiply by n. This completes the bound on T_1 subject to the dictionary growth constraint.

Bound on T_2 The term T_2 is the bias incurred by the thinning operation. We will tune the budget schedule $\{\epsilon_i\}$ so that T_2 tends to 0 at the same rate as T_1. The goal is to keep epsilon as large as possible in order to retain as few points as possible while preserving the rate of posterior contraction. Noticeably, we do not need this term to converge any faster than $n\log n/f(n)^2$. The only dependence on the point stream $\mathbf{S}$ comes from $|\mathbf{D}_j|$, and we directly bound this term by observing that $|\mathbf{D}_j| \leq j$ so

$$\frac{|\mathbf{D}_j|}{|\mathbf{D}_{j+1}|} \leq \frac{j}{j+1}.$$

Therefore, returning to T_2 in (5.30), we have

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{n}\epsilon_i\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] &\leq \sum_{i=1}^{n}\epsilon_i\left(\prod_{j=i}^{n-1}\frac{j}{j+1}\right)^2 \\ &= \sum_{i=1}^{n}\epsilon_i\frac{i^2}{n^2} \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\epsilon_i i^2 \end{aligned} \tag{5.35}$$

Therefore the goal is to choose ϵ_i satisfying (up to generic constants)

$$\begin{aligned} &\frac{1}{n^2}\sum_{i=1}^{n}\epsilon_i i^2 \leq \frac{n\log n}{f(n)^2} \\ \iff &\sum_{i=1}^{n}\epsilon_i i^2 \leq \frac{n^3\log n}{f(n)^2} \end{aligned} \tag{5.36}$$

The first line above states that T_1 and T_2 converge at the same rate. The second line simply multiplies the first inequality by n^2. Therefore, to satisfy the condition on the right-hand side of the previous expression, one can select ϵ_i according to the growth condition associated with the posterior contraction rate of the sampled process, i.e., the upper bound on T_1 in (5.50). According to this rate, the model complexity increases at least according to the growth rate $f(i)$. We set $\epsilon_i = \log(i)/f(i)^2$ and demonstrate that (5.36) is met:

$$\begin{aligned} \sum_{i=1}^{n}\epsilon_i i^2 &= \sum_{i=1}^{n}\frac{i^2\log(i)}{f(i)^2} \\ &\leq \sum_{i=1}^{n}\frac{n^2\log(n)}{f(n)^2} \\ &= \frac{n^3\log(n)}{f(n)^2} \end{aligned} \tag{5.37}$$

The first line is the evaluation of ϵ_i. The inequality in the second line holds because $f(i) \leq i$ so the summand is increasing in i. After we remove the dependence on the index i we multiply by the maximum summation index n to achieve the desired result. This concludes the bound of T_2.

Finally we can substitute the bound for T_1 ((5.34)) and the bound for T_2 ((5.37)) into (5.30) to get

$$\mathbb{E}\left[\sum_{i=1}^{n}\left(\frac{S_i^2 + r_i\|h_i\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{i-1}|+1)^2} + \epsilon_i\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] \leq C\frac{n\log(n)}{f(n)^2}. \tag{5.38}$$

Finally we revisit (5.29) to achieve our desired conclusion:

$$\begin{aligned}\mathbb{E}\left[\mathrm{KSD}(q_{\mathbf{D}_n})^2\right] &\leq \mathbb{E}\left[\exp\left(\sum_{j=1}^{n}\frac{1}{r_j}\right)\sum_{i=1}^{n}\left(\frac{S_i^2+r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{i-1}|+1\right)^2}+\epsilon_i\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] \\ &= \mathbb{E}\left[e\sum_{i=1}^{n}\left(\frac{S_i^2+r_i\|h_i\|_{\mathcal{K}_0}^2}{\left(|\mathbf{D}_{i-1}|+1\right)^2}+\epsilon_i\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] \\ &\leq C\frac{n\log(n)}{f(n)^2}\end{aligned} \tag{5.39}$$

The equality in the second line follows from the fact that $r_j = n$ for all j. The final inequality follows from an application of (5.38) to reach the desired conclusion of Theorem 5.7.1. ■

Corollary 5.7.1 (i.i.d. Convergence with constant thinning budget) *Fix the desired KSD convergence radius Δ. Assume that the dictionary model order growth rate f takes the parametric form $f(i) = \sqrt{i^{1+\alpha}\log(i)}$ with $1 < \alpha < 2$. With constant thinning budget $\epsilon = \mathcal{O}\left(\Delta^{1+\frac{1}{\alpha}}\right)$, after $n = \mathcal{O}\left(\frac{1}{\Delta^{\frac{1}{\alpha}}}\right)$ steps the KSD satisfies*

$$\mathbb{E}\left[KSD\left(q_{\mathbf{D}_n}\right)^2\right] \leq C\Delta \tag{5.40}$$

for some generic constant C. Equivalently, if we fix a constant thinning budget ϵ, after $n = \mathcal{O}\left(\epsilon^{-\frac{2}{\alpha+1}}\right)$ steps the KSD satisfies

$$\mathbb{E}\left[KSD\left(q_{\mathbf{D}_n}\right)^2\right] \leq C\epsilon^{\frac{2}{1+\frac{1}{\alpha}}} \tag{5.41}$$

Proof We will revisit terms T_1 and T_2 in (5.30) from the proof of Theorem 5.7.1 and show how to obtain the desired convergence rate in the constant thinning budget regime.

First, (5.34) demonstrates that the sampling error term T_1 converges at a rate of $\frac{n\log(n)}{f(n)^2}$ (up to generic constants). We set the convergence rate equal to the convergence radius and apply algebraic manipulations to derive the number of sampling and thinning iterations required. In

particular, write:

$$\Delta = \frac{n\log(n)}{f(n)^2} = \frac{n\log(n)}{n^{1+\alpha}\log(n)} = \frac{1}{n^\alpha} \tag{5.42}$$

Since the convergence rate $\frac{n\log(n)}{f(n)^2}$ holds up to generic constants we can conclude that after $n = \mathcal{O}\left(\frac{1}{\Delta^{\frac{1}{\alpha}}}\right)$ steps $T_1 \leq C\Delta$.

Next we bound T_2, the error associated with memory-reduction. We demonstrated in the proof of Theorem 5.7.1 that

$$T_2 \leq \frac{1}{n^2}\sum_{i=1}^{n} \epsilon_i i^2. \tag{5.43}$$

Since $\epsilon_i = \epsilon = \mathcal{O}\left(\Delta^{1+\frac{1}{\alpha}}\right)$ is constant with respect to the summation index i, we can evaluate the expression above as

$$\begin{aligned}
\frac{1}{n^2}\sum_{i=1}^{n} \epsilon_i i^2 &= \frac{\Delta^{1+\frac{1}{\alpha}}}{n^2}\sum_{i=1}^{n} i^2 \\
&\leq \frac{\Delta^{1+\frac{1}{\alpha}}}{n^2} Cn^3 \\
&= Cn\Delta^{1+\frac{1}{\alpha}} \\
&= C\frac{\Delta^{1+\frac{1}{\alpha}}}{\Delta^{\frac{1}{\alpha}}} = C\Delta
\end{aligned} \tag{5.44}$$

In the first line we plug in ϵ. In the second line we make use of the general summation formula $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(n+2)}{6} \leq Cn^3$. In the final line we apply the fact that $n = \mathcal{O}\left(\frac{1}{\Delta^{\frac{1}{\alpha}}}\right)$ and conclude that $T_2 \leq C\Delta$ for some generic constant C.

Therefore we have $T_1 + T_2 \leq C\Delta$ which is the desired conclusion. ■

Next we consider the case when samples are generated using any MCMC sampling procedure. The one constraint we place on the sampler is that it exhibits geometric convergence with respect to a given function V. The function V, which controls the rate of geometric convergence, will contribute to the sampling errror term T_1 in our analysis.

Definition 5.7.1 *A Markov chain $\{\mathbf{y}_n\} \subset \mathcal{X}$ with n-th step transition kernel P^n is V-***uniformly**

ergodic *for a positive function* $V : \mathcal{X} \to [1, \infty]$ *if there exists* $R \in [0, \infty], \rho \in (0, 1)$ *such that*

$$\sup_{\mathbf{x} \in X} \frac{\|P^n(\mathbf{y}, \mathbf{x}) - p(\mathbf{x})\|}{V(\mathbf{x})} \leq RV(\mathbf{y})p^n \tag{5.45}$$

We introduce new notation from [91] by defining the function $V_{\pm}$:

$$V_{\pm}(s) = \sup_{\mathbf{x}: k_0(\mathbf{x},\mathbf{x}) \leq s^2} k_0(\mathbf{x}, \mathbf{x})^{1/2} V(x)^{\pm 1} \tag{5.46}$$

We will use these two functions $V_{\pm}$ and a result from [91] to control the sampling error of a V-uniformly ergodic Markov chain. First we extend the convergence guarantees of Algorithm 2 to general V-uniformly ergodic samplers in Theorem 5.7.2. Then, when MALA is V-uniformly ergodic, the desired result follows.

Theorem 5.7.2 (V-Uniformly Ergodic MCMC Sampling and Decaying Budget) *Suppose the same conditions as Lemma 5.7.2, and the Stein kernel* k_0 *satisfies* $\mathbb{E}_{\mathbf{y} \sim P}\left[e^{\gamma k_0(\mathbf{y},\mathbf{y})}\right] = b < \infty$*, and the lower-bound on the dictionary growth condition* $|\mathbf{D}_i| \geq Cf(i)$ *holds where* $f(i) = o\left(\sqrt{i \log(i)}\right)$*, with compression budgets set as* $\epsilon_i = \frac{\log(i)}{f(i)^2}$*. Further assume that the candidate samples in each* $\{\mathbf{y}_{i,l}^{m_i}\}$ *are produced by a V-uniformly ergodic markov chain. Then the iterate* $\mathbf{D}_n = KSDT(\mathbf{D}_{n-1} \cup \{\mathbf{x}_n\}, \epsilon_n)$ *satisfies*

$$\mathbb{E}\left[KSD(q_{\mathbf{D}_n})^2\right] \leq C\left(\frac{n \log(n)}{f(n)^2} + \frac{1}{nf(n)^2} \sum_{i=1}^{n} V_+(S_i)V_-(S_i)\right) \tag{5.47}$$

Proof This proof is similar to the proof of Theorem 5.7.1. It differs only in the bound applied to the term $\|h_j\|_{\mathcal{K}_0}^2$ which is a subcomponent of the sampling bias term T_1. This difference is due to the non-i.i.d. sampling procedure used to generate candidate samples that in turn define h_j. When the candidate samples $\{\mathbf{y}_{i,l}\}_{l=1}^{m_i}$ are generated from a V-uniformly ergodic markov chain, we can apply the following bound from [91][Appendix A.2, Equation (22)]:

$$\mathbb{E}\left[\|h_j\|_{\mathcal{K}_0}^2\right] \leq \frac{4b}{\gamma} \exp\left(-\frac{\gamma}{2} S_j^2\right) + 2RV_+(S_j)V_-(S_j)\left(\frac{1+2p}{1-p}\right)\frac{1}{m_j} \tag{5.48}$$

where the constants p and R come from the V-uniform ergodicity of the Markov chain that generates the candidate samples.

Our goal is to mimic the conclusion of (5.33) in our current setting.

$$\mathbb{E}\left[\sum_{i=1}^{n}\left(\frac{S_i^2+r_i\|h_i\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{i-1}|+1)^2}\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] \leq C\frac{1}{f(n)^2}\sum_{i=1}^{n}\left(S_i^2+r_i\exp\left(-\frac{\gamma}{2}S_i^2\right)+\frac{V_+(S_i)V_-(S_i)}{m_j}\right) \tag{5.49}$$

The first inequality is an application of (5.48) to the final conclusion of (5.31) from the proof of Theorem 5.7.1. We also absorb the constant factors into C. We select $S_i=\sqrt{\frac{2}{\gamma}\log(n\wedge m_i)}$, $r_i=n$, and $m_i=n$ to achieve the bound

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{n}\left(\frac{S_i^2+r_i\|h_i\|_{\mathcal{K}_0}^2}{(|\mathbf{D}_{i-1}|+1)^2}\right)\left(\prod_{j=i}^{n-1}\frac{|\mathbf{D}_j|}{|\mathbf{D}_j|+1}\right)^2\right] &\leq C\frac{1}{f(n)^2}\sum_{i=1}^{n}\left(\log(n)+\frac{V_+(S_i)V_-(S_i)}{n}\right)\\ &\leq C\left(\frac{n\log(n)}{f(n)^2}+\frac{1}{nf(n)^2}\sum_{i=1}^{n}V_+(S_i)V_-(S_i)\right)\end{aligned} \tag{5.50}$$

The first inequality is the conclusion of the previous equation with S_i, and r_i evaluated and all constants absorbed. The second inequality holds because the summand does not depend on the index so we can multiply by n. This completes the bound on T_1 in the setting where samples are generated by a V-uniformly ergodic Markov chain.

The bound on T_2 in Theorem 5.7.1 does not depend on the sample generation mechanism, so we can draw the same conclusion as (5.37) which does not introduce any terms with higher asymptotic growth order than $n\log(n)/f(n)^2$. Therefore the bound from (5.50) is the asymptotic upper bound, and we have achieved the desired result. ■

Our goal is to show that using MALA to generate samples ensures convergence of Algorithm 2. When MALA is inwardly convergent, MALA is V-uniformly ergodic. We make use of this

fact to add one more condition (inwardly convergent) and present a convergence result for Algorithm 2 using the MALA sampler.

Corollary 5.7.2 (MALA Convergence with Decaying Budget) *Assume that the candidate sample sets* $\{\mathbf{y}_{i,l}\}_{l=1}^{m_i}$ *are generated by the Metropolis-Adjusted Langevin Algorithm (MALA) transition kernel with stepsize* h *and that MALA is inwardly convergent for the target density* p*. Further assume that* p *is distantly dissipative. Then the iterates* $\mathbf{D}_n$ *of Algorithm 2 satisfy*

$$\mathbb{E}\left[KSD(q_{\mathbf{D}_n})^2\right] \leq C\left(\frac{n\log(n)}{f(n)^2}\right)$$

Proof We make use of two results from [91], Appendix A.3. First we use the fact that MALA is inwardly convergent to conclude that MALA is V-uniformly ergodic with $V(\mathbf{x}) = 1 + \|x\|_2$. The second result is the set of bounds $V_+(s) \leq C(s + s^2)$ and $V_-(s) \leq C$ for some generic constant C. Then we can plug in $S_i = \sqrt{\frac{2}{\gamma}\log(n)}$ (since $m_i = n$) to bound the second term in Equation 5.50:

$$\begin{aligned}
&\frac{1}{nf(n)^2}\sum_{i=1}^{n} V_+(S_i)V_-(S_i) \\
&\qquad \leq \frac{C}{nf(n)^2}\sum_{i=1}^{n}\left(S_i + S_i^2\right) \\
&\qquad \leq \frac{C}{nf(n)^2}\sum_{i=1}^{n}\left(\sqrt{\log(n)} + \log(n)\right) \\
&\qquad \leq \frac{C\sqrt{log(n)} + \log(n)}{f(n)^2} \\
&\qquad \leq \frac{C\log(n)}{f(n)^2}
\end{aligned} \tag{5.51}$$

The first inequality is result of the evaluation of $V_\pm(S_i)$ and the absorption of constants. The second inquality is the results of the evaluation of $S_i = \sqrt{\frac{2}{\gamma}\log(n)}$ and the absorption of constants. The third inequality follows from the fact that the summand does not depend on the index. The final inequality is an application of the fact that $\sqrt{\log(n)} \leq \log(n)$ for $n \geq 10$, which we assume as a reasonable minimum number of steps.

Since the final term in (5.51) is asymptotically upper bounded by $n\log(n)/f(n)^2$ due to the extra factor of n we can ignore this term in Equation 5.47 and draw the desired conclusion that

$$\mathbb{E}\left[\mathrm{KSD}(q_{\mathbf{D}_n})^2\right] \leq C\left(\frac{n\log(n)}{f(n)^2}\right) \tag{5.52}$$

■

Corollary 5.7.3 (MALA Convergence with constant thinning budget) *Fix the desired KSD convergence radius Δ. Assume that the dictionary model order growth rate f takes the parametric form $f(i) = \sqrt{i^{1+\alpha}\log(i)}$ with $1 < \alpha < 2$. With constant thinning budget $\epsilon = \mathcal{O}\left(\Delta^{1+\frac{1}{\alpha}}\right)$, after $n = \mathcal{O}\left(\frac{1}{\Delta^{\frac{1}{\alpha}}}\right)$ steps the KSD satisfies*

$$\mathbb{E}\left[KSD\left(q_{\mathbf{D}_n}\right)^2\right] \leq c\Delta \tag{5.53}$$

for some generic constant C. Equivalently, if we fix a constant thinning budget ϵ, after $n = \mathcal{O}\left(\epsilon^{-\frac{2}{\alpha+1}}\right)$ steps the KSD satisfies

$$\mathbb{E}\left[KSD\left(q_{\mathbf{D}_n}\right)^2\right] \leq C\epsilon^{\frac{2}{1+\frac{1}{\alpha}}} \tag{5.54}$$

This proof is exactly the same as the proof of Corollary 5.7.1 since the asymptotic convergence rate achieved in Corollary 5.7.2 is exactly the same as in the i.i.d. case in Theorem 5.7.1. We note that the convergence rate from Corollary 5.7.3 also trivially extends to the general case of V-uniformly ergodic samplers using the result of Theorem 5.7.2 and the analysis of Corollary 5.3.1.

Chapter 6

Conclusion

6.1 Summary

In this dissertation we developed several algorithms that advance the compressed training capabilities of low-rank tensor models and the uncertainty quantification capabilities of general compressed nonparametric models. Our initial motivating problem was compressed training with further compression during training for edge-device machine learning. This led to the development of our initial compressed low-rank tensorized training with rank reduction developed in Chapter 3. In the followings chapters we addressed several new challenges raised by the initial work in Chapter 3. First, in Chapter 4 we extended the compressed training with rank reduction to multiple tensor formats and developed a scalable algorithm suitable for general-purpose rank reduction in nonlinear tensor problems. Next, in Chapter 5 we revisited a challenge raised in Chapter 3, selecting the number of particles in a nonparametric representation, by developing an algorithm to automatically determine the appropriate complexity of nonparametric distributional representations.

6.2 Future Directions

Hardware Acceleration of Tensor Methods Low-rank tensor methods consistently provide memory savings over uncompressed models. However savings in computation require careful consideration of tensor reshaping dimensions, tensor format, and tensor contraction

order [96, 146]. Hardware acceleration of low-rank tensor-compressed deep learning models is another active area of research [18, 19]. Different tensor formats and reshaping schemes allow one to determine the memory/FLOPs tradeoff of different tensor compression approaches. Tailoring these compression schemes to different hardware platforms via techniques such as hardware-aware neural architecture search [147] and tensor architecture search [148] is an open challenge.

Compressed Bayesian Learning Compressed nonparametric distributional representations have many applications, allowing the work presented in Chapter 5 to serve as an "inner loop" in a range of tasks. Two tasks of interest are batch Bayesian optimization and energy-based modeling. Recent work [149] reformulates batch Bayesian optimization as particle optimization. Thinning methods based on the KSD may permit batch size selection for experimental design. In energy-based modeling MCMC samplers are a key component of gradient estimation in popular approaches [150]. More generally, the incorporation of KSD-based samplers with the Involutive MCMC framework [151] provides many avenues for exploration.

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[3] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, *et. al.*, *Deep learning recommendation model for personalization and recommendation systems*, *arXiv:1906.00091* (2019).

[4] Y. LeCun, *The MNIST database of handwritten digits*, *http://yann. lecun. com/exdb/mnist/* (1998).

[5] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, *arXiv preprint arXiv:1409.1556* (2014).

[6] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, *Neural computation* **9** (1997), no. 8 1735–1780.

[7] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding by generative pre-training*, .

[8] S. Hochreiter, *The vanishing gradient problem during learning recurrent neural nets and problem solutions*, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **6** (1998), no. 02 107–116.

[9] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in *Proc. International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[11] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, *Image transformer*, in *International Conference on Machine Learning*, pp. 4055–4064, PMLR, 2018.

[12] L. Dong, S. Xu, and B. Xu, *Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition*, in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5884–5888, IEEE, 2018.

[13] "Ai and the memory wall." `https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8`. Accessed: 2022-01-02.

[14] T. Chen, L. Li, and Y. Sun, *Differentiable product quantization for end-to-end embedding compression*, in *International Conference on Machine Learning*, pp. 1617–1626, PMLR, 2020.

[15] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et. al.*, *In-datacenter performance analysis of a tensor processing unit*, in *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

[16] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, *Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks*, *IEEE journal of solid-state circuits* **52** (2016), no. 1 127–138.

[17] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, *Hardware for machine learning: Challenges and opportunities*, in *IEEE Custom Integrated Circuits Conference*, pp. 1–8, 2017.

[18] K. Zhang, C. Hawkins, X. Zhang, C. Hao, and Z. Zhang, *On-FPGA training with ultra memory reduction: A low-precision tensor method*, *ICLR Workshop of Hardware Aware Efficient Training* (May 2021).

[19] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, *Tie: energy-efficient tensor train-based inference engine for deep neural network*, in *Proc. Int. Symp. Computer Arch.*, pp. 264–278, 2019.

[20] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, *How to evaluate deep neural network processors: Tops/w (alone) considered harmful*, *IEEE Solid-State Circuits Magazine* **12** (2020), no. 3 28–41.

[21] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, *Eie: Efficient inference engine on compressed deep neural network*, *ACM SIGARCH Computer Architecture News* **44** (2016), no. 3 243–254.

[22] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, *Beyond inferring class representatives: User-level privacy leakage from federated learning*, in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2512–2520, IEEE, 2019.

[23] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, *Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges*, *Information fusion* **58** (2020) 52–68.

[24] B. Lakshminarayanan, A. Pritzel, and C. Blundell, *Simple and scalable predictive uncertainty estimation using deep ensembles*, *arXiv preprint arXiv:1612.01474* (2016).

[25] Y. Gal and Z. Ghahramani, *Dropout as a Bayesian approximation: Representing model uncertainty in deep learning*, in *International Conference on Machine Learning*, pp. 1050–1059, 2016.

[26] A. Kendall and Y. Gal, *What uncertainties do we need in Bayesian deep learning for computer vision?*, in *Advances in neural information processing systems*, pp. 5574–5584, 2017.

[27] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, *Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights*, *Proceedings of the IEEE* **109** (2021), no. 10 1706–1752.

[28] Y. LeCun, J. S. Denker, and S. A. Solla, *Optimal brain damage*, in *Advances in neural information processing systems*, pp. 598–605, 1990.

[29] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, *arXiv:1510.00149* (2015).

[30] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, *What is the state of neural network pruning?*, *arXiv preprint arXiv:2003.03033* (2020).

[31] M. Zhu and S. Gupta, *To prune, or not to prune: exploring the efficacy of pruning for model compression*, *arXiv preprint arXiv:1710.01878* (2017).

[32] C. Wang, G. Zhang, and R. Grosse, *Picking winning tickets before training by preserving gradient flow*, *arXiv preprint arXiv:2002.07376* (2020).

[33] J. van Amersfoort, M. Alizadeh, S. Farquhar, N. Lane, and Y. Gal, *Single shot structured pruning before training*, *arXiv preprint arXiv:2007.00389* (2020).

[34] J. Frankle and M. Carbin, *The lottery ticket hypothesis: Finding sparse, trainable neural networks*, *arXiv preprint arXiv:1803.03635* (2018).

[35] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, *Stabilizing the lottery ticket hypothesis*, *arXiv preprint arXiv:1903.01611* (2019).

[36] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, and M. Carbin, *The lottery ticket hypothesis for pre-trained bert networks*, *arXiv preprint arXiv:2007.12223* (2020).

[37] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, *Model compression and hardware acceleration for neural networks: A comprehensive survey*, *Proceedings of the IEEE* **108** (2020), no. 4 485–532.

[38] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, *Deep learning with limited numerical precision*, in *International Conference on Machine Learning*, pp. 1737–1746, 2015.

[39] G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson, and C. De Sa, *Swalp: Stochastic weight averaging in low precision training*, in *International Conference on Machine Learning*, pp. 7015–7024, PMLR, 2019.

[40] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, *Haq: Hardware-aware automated quantization with mixed precision*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8612–8620, 2019.

[41] Y. Fu, H. Guo, M. Li, X. Yang, Y. Ding, V. Chandra, and Y. Lin, *Cpt: Efficient deep neural network training via cyclic precision*, *arXiv preprint arXiv:2101.09868* (2021).

[42] M. Courbariaux, Y. Bengio, and J.-P. David, *Binaryconnect: Training deep neural networks with binary weights during propagations*, in *Advances in neural information processing systems*, pp. 3123–3131, 2015.

[43] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, *arXiv:1503.02531* (2015).

[44] J. H. Cho and B. Hariharan, *On the efficacy of knowledge distillation*, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4794–4802, 2019.

[45] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, *Exploiting linear structure within convolutional networks for efficient evaluation*, in *Advances in neural information processing systems*, pp. 1269–1277, 2014.

[46] M. Jaderberg, A. Vedaldi, and A. Zisserman, *Speeding up convolutional neural networks with low rank expansions*, *arXiv preprint arXiv:1405.3866* (2014).

[47] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, *Speeding-up convolutional neural networks using fine-tuned cp-decomposition*, in *International Conference on Learning Representations*, 2015.

[48] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, *Compression of deep convolutional neural networks for fast and low power mobile applications*, *arXiv preprint arXiv:1511.06530* (2015).

[49] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, *SIAM review* **51** (2009), no. 3 455–500.

[50] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, *Tensorizing neural networks*, in *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

[51] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, *Ultimate tensorization: compressing convolutional and FC layers alike*, *arXiv preprint arXiv:1611.03214* (2016).

[52] G. G. Calvi, A. Moniri, M. Mahfouz, Z. Yu, Q. Zhao, and D. P. Mandic, *Tucker tensor layer in fully connected neural networks*, *arXiv preprint arXiv:1903.06133* (2019).

[53] B. Recht, M. Fazel, and P. A. Parrilo, *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization*, *SIAM review* **52** (2010), no. 3 471–501.

[54] S. Gandy, B. Recht, and I. Yamada, *Tensor completion and low-n-rank tensor recovery via convex optimization*, *Inverse Problems* **27** (2011), no. 2.

[55] J. Liu, P. Musialski, P. Wonka, and J. Ye, *Tensor completion for estimating missing values in visual data*, *IEEE Trans. Pattern Analysis and Machine Intelligence* **35** (2013), no. 1 208–220.

[56] M. Imaizumi, T. Maehara, and K. Hayashi, *On tensor train rank minimization: Statistical efficiency and scalable algorithm*, in *Advances in Neural Information Processing Systems*, pp. 3930–3939, 2017.

[57] J. A. Bazerque, G. Mateos, and G. B. Giannakis, *Rank regularization and Bayesian inference for tensor completion and extrapolation*, *IEEE transactions on signal processing* **61** (2013), no. 22 5689–5703.

[58] Q. Zhao, L. Zhang, and A. Cichocki, *Bayesian CP factorization of incomplete tensors with automatic rank determination*, *IEEE Trans. Pattern Analysis and Machine Intelligence* **37** (2015), no. 9 1751–1763.

[59] P. Rai, Y. Wang, S. Guo, G. Chen, D. Dunson, and L. Carin, *Scalable Bayesian low-rank decomposition of incomplete multiway tensors*, in *International Conference on Machine Learning*, 2014.

[60] R. Guhaniyogi, S. Qamar, and D. B. Dunson, *Bayesian tensor regression*, *The Journal of Machine Learning Research* **18** (2017), no. 1 2733–2763.

[61] C. Hawkins and Z. Zhang, *Robust factorization and completion of streaming tensor data via variational Bayesian inference*, *arXiv preprint arXiv:1809.01265* (2018).

[62] Q. Zhao, G. Zhou, L. Zhang, A. Cichocki, and S.-I. Amari, *Bayesian robust tensor factorization for incomplete multiway data*, *IEEE Trans. Neural Networks and Learning Systems* **27** (2016), no. 4 736–748.

[63] H. Zhou, L. Li, and H. Zhu, *Tensor regression with applications in neuroimaging data analysis*, *Journal of the American Statistical Association* **108** (2013), no. 502 540–552.

[64] D. J. MacKay, *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

[65] R. M. Neal, *Bayesian learning via stochastic dynamics*, in *NIPS*, pp. 475–482, 1993.

[66] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[67] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, *On calibration of modern neural networks*, in *International Conference on Machine Learning*, pp. 1321–1330, PMLR, 2017.

[68] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. Wilson, *What are bayesian neural network posteriors really like?*, *arXiv preprint arXiv:2104.14421* (2021).

[69] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, *Stochastic variational inference*, *The Journal of Machine Learning Research* **14** (2013), no. 1 1303–1347.

[70] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, *Weight uncertainty in neural network*, in *International Conference on Machine Learning*, pp. 1613–1622, PMLR, 2015.

[71] J. M. Alvarez and M. Salzmann, *Compression-aware training of deep networks*, in *Advances in Neural Information Processing Systems*, pp. 856–867, 2017.

[72] S. J. Hanson and L. Y. Pratt, *Comparing biases for minimal network construction with back-propagation*, in *Advances in neural information processing systems*, pp. 177–185, 1989.

[73] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, *Structured Bayesian pruning via log-normal multiplicative noise*, in *NIPS*, pp. 6775–6784, 2017.

[74] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, *Incremental network quantization: Towards lossless cnns with low-precision weights*, *arXiv:1702.03044* (2017).

[75] D. Yang, W. Yu, H. Mu, and G. Yao, *Dynamic programming assisted quantization approaches for compressing normal and robust DNN models*, in *Proc. Asia and South Pacific Design Automation Conference*, pp. 351–357, 2021.

[76] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*, in *ICASSP*, pp. 6655–6659, 2013.

[77] J. Xue, J. Li, and Y. Gong, *Restructuring of deep neural network acoustic models with singular value decomposition.*, in *Interspeech*, pp. 2365–2369, 2013.

[78] Y. Ma, R. Chen, W. Li, F. Shang, W. Yu, M. Cho, and B. Yu, *A unified approximation framework for compressing and accelerating deep neural networks*, in *International Conf. on Tools with Artificial Intelligence*, pp. 376–383, 2019.

[79] Z. He, S. Gao, L. Xiao, D. Liu, H. He, and D. Barber, *Wider and deeper, cheaper and faster: Tensorized LSTMs for sequence learning*, *NIPS* **30** (2017) 1–11.

[80] C. Cui, C. Hawkins, and Z. Zhang, *Tensor methods for generating compact uncertainty quantification and deep learning models*, in *Intl. Conf. Computer-Aided Design*, pp. 1–6, 2019.

[81] X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, M. Zhou, and D. Song, *A tensorized transformer for language modeling*, in *NIPS*, pp. 2232–2242, 2019.

[82] K. Zhang, X. Zhang, and Z. Zhang, *Tucker tensor decomposition on FPGA*, in *Proc. Intl. Conf. Computer-Aided Design*, pp. 1–8, 2019.

[83] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, *Learning structured sparsity in deep neural networks*, *NIPS* **29** (2016) 2074–2082.

[84] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *Quantized neural networks: Training neural networks with low precision weights and activations*, *The Journal of Machine Learning Research* **18** (2017), no. 1 6869–6898.

[85] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof, *et. al.*, *Flexpoint: An adaptive numerical format for efficient training of deep neural networks*, in *NIPS*, pp. 1742–1752, 2017.

[86] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, *Ultra-low precision 4-bit training of deep neural networks*, *NIPS* **33** (2020).

[87] I. V. Oseledets, *Tensor-train decomposition*, *SIAM J. Scientific Computing* **33** (2011), no. 5 2295–2317.

[88] Q. Liu and D. Wang, *Stein variational gradient descent: A general purpose Bayesian inference algorithm*, in *Proc. Advances In Neural Information Processing Systems*, pp. 2378–2386, 2016.

[89] J. Gusak, M. Kholiavchenko, E. Ponomarev, L. Markeeva, I. Oseledets, and A. Cichocki, *Musco: Multi-stage compression of neural networks*, *arXiv preprint arXiv:1903.09973* (2019).

[90] K. Zhang, C. Hawkins, X. Zhang, and Z. Zhang, *Low-precision and rank-adaptive training of tensorized neural networks*, in *IEEE/ACM International Conf. Computer-Aided Design*, 2020 (to appear).

[91] W. Y. Chen, A. Barp, F.-X. Briol, J. Gorham, M. Girolami, L. Mackey, C. Oates, *et. al.*, *Stein point markov chain monte carlo*, *arXiv preprint arXiv:1905.03673* (2019).

[92] Q. Liu, J. Lee, and M. Jordan, *A kernelized stein discrepancy for goodness-of-fit tests*, in *International conference on machine learning*, pp. 276–284, PMLR, 2016.

[93] J. D. Carroll and J.-J. Chang, *Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition*, *Psychometrika* **35** (1970), no. 3 283–319.

[94] R. A. Harshman, M. E. Lundy, *et. al.*, *PARAFAC: Parallel factor analysis*, *Computational Statistics and Data Analysis* **18** (1994), no. 1 39–72.

[95] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, *Psychometrika* **31** (1966), no. 3 279–311.

[96] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, *Wide compression: Tensor ring nets*, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9329–9338, 2018.

[97] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, *arXiv:1412.6980* (2014).

[98] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, *arXiv preprint arXiv:1708.07747* (2017).

[99] A. Krizhevsky, V. Nair, and G. Hinton, *The CIFAR-10 dataset*, *online: http://www. cs. toronto. edu/kriz/cifar. html* (2014).

[100] F. Chollet *et. al.*, "Keras." `https://github.com/fchollet/keras`, 2015.

[101] D. Wang, Z. Zeng, and Q. Liu, *Stein variational message passing for continuous graphical models*, in *International Conference on Machine Learning*, pp. 5219–5227, PMLR, 2018.

[102] M. Ye, T. Ren, and Q. Liu, *Stein self-repulsive dynamics: Benefits from past samples*, in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 241–252, Curran Associates, Inc., 2020.

[103] V. Khrulkov, O. Hrinchuk, L. Mirvakhabova, and I. Oseledets, *Tensorized embedding layers for efficient model compression*, *arXiv:1901.10787* (2019).

[104] C. Hawkins and Z. Zhang, *Variational Bayesian inference for robust streaming tensor factorization and completion*, in *2018 IEEE International Conference on Data Mining*, pp. 1446–1451, IEEE, 2018.

[105] C. Hawkins and Z. Zhang, *Bayesian tensorized neural networks with automatic rank selection*, *arXiv:1905.10478* (2019).

[106] C. M. Carvalho, N. G. Polson, and J. G. Scott, *The horseshoe estimator for sparse signals*, *Biometrika* **97** (2010), no. 2 465–480.

[107] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, *An introduction to variational methods for graphical models*, *Machine learning* **37** (1999), no. 2 183–233.

[108] M. P. Wand, J. T. Ormerod, S. A. Padoan, R. Frühwirth, *et. al.*, *Mean field variational Bayes for elaborate distributions*, *Bayesian Analysis* **6** (2011), no. 4 847–900.

[109] S. Ghosh, J. Yao, and F. Doshi-Velez, *Model selection in Bayesian neural networks via horseshoe priors*, *Journal of Machine Learning Research* **20** (2019), no. 182 1–46.

[110] S. Nakajima and M. Sugiyama, *Analysis of empirical MAP and empirical partially Bayes: Can they be alternatives to variational Bayes?*, in *Artificial Intelligence and Statistics*, pp. 20–28, 2014.

[111] B. Lakshminarayanan, A. Pritzel, and C. Blundell, *Simple and scalable predictive uncertainty estimation using deep ensembles*, in *NIPS*, pp. 6402–6413, 2017.

[112] P. G. Constantine, C. Kent, and T. Bui-Thanh, *Accelerating markov chain monte carlo with active subspaces*, *SIAM Journal on Scientific Computing* **38** (2016), no. 5 A2779–A2805.

[113] B. Peherstorfer, K. Willcox, and M. Gunzburger, *Survey of multifidelity methods in uncertainty propagation, inference, and optimization*, *Siam Review* **60** (2018), no. 3 550–591.

[114] J. Martin, L. C. Wilcox, C. Burstedde, and O. Ghattas, *A stochastic newton mcmc method for large-scale statistical inverse problems with application to seismic inversion*, *SIAM Journal on Scientific Computing* **34** (2012), no. 3 A1460–A1487.

[115] M. Riabiz, W. Chen, J. Cockayne, P. Swietach, S. A. Niederer, L. Mackey, C. Oates, *et. al.*, *Optimal thinning of mcmc output*, *arXiv preprint arXiv:2005.03952* (2020).

[116] O. Teymur, J. Gorham, M. Riabiz, C. Oates, *et. al.*, *Optimal quantisation of probability measures using maximum mean discrepancy*, *arXiv preprint arXiv:2010.07064* (2020).

[117] M. Welling and Y. W. Teh, *Bayesian learning via stochastic gradient langevin dynamics*, in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 681–688, Citeseer, 2011.

[118] Y.-A. Ma, T. Chen, and E. B. Fox, *A complete recipe for stochastic gradient mcmc*, *arXiv preprint arXiv:1506.04696* (2015).

[119] C. Stein, *A bound for the error in the normal approximation to the distribution of a sum of dependent random variables*, in *Proceedings of the sixth Berkeley symposium on mathematical statistics and probability, volume 2: Probability theory*, pp. 583–602, University of California Press, 1972.

[120] A. Berlinet and C. Thomas-Agnan, *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[121] B. K. Sriperumbudur, A. Gretton, K. Fukumizu, B. Schölkopf, and G. R. Lanckriet, *Hilbert space embeddings and metrics on probability measures*, *The Journal of Machine Learning Research* **11** (2010) 1517–1561.

[122] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, *A kernel two-sample test*, *The Journal of Machine Learning Research* **13** (2012), no. 1 723–773.

[123] T. Campbell and T. Broderick, *Bayesian coreset construction via greedy iterative geodesic ascent*, in *International Conference on Machine Learning*, pp. 698–706, PMLR, 2018.

[124] T. Campbell and T. Broderick, *Automated scalable bayesian inference via hilbert coresets*, *The Journal of Machine Learning Research* **20** (2019), no. 1 551–588.

[125] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA, 2006.

[126] T. Hofmann, B. Schölkopf, and A. J. Smola, *Kernel methods in machine learning*, *The annals of statistics* **36** (2008), no. 3 1171–1220.

[127] C. Williams and M. Seeger, *Using the nyström method to speed up kernel machines*, in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2001.

[128] M. W. Seeger, C. K. Williams, and N. D. Lawrence, *Fast forward selection to speed up sparse gaussian process regression*, in *International Workshop on Artificial Intelligence and Statistics*, pp. 254–261, PMLR, 2003.

[129] Z. Wang, K. Crammer, and S. Vucetic, *Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training*, *Journal of Machine Learning Research* **13** (2012), no. 100 3103–3131.

[130] E. Snelson and Z. Ghahramani, *Sparse gaussian processes using pseudo-inputs*, *Advances in Neural Information Processing Systems* **18** (2005) 1257–1264.

[131] A. Koppel, G. Warnell, E. Stump, and A. Ribeiro, *Parsimonious online learning with kernels via sparse projections in function space*, in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4671–4675, IEEE, 2017.

[132] A. Koppel, H. Pradhan, and K. Rajawat, *Consistent online gaussian process regression without the sample complexity bottleneck*, *arXiv preprint arXiv:2004.11094* (2020).

[133] J. Gorham and L. Mackey, *Measuring sample quality with kernels*, in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1292–1301, JMLR. org, 2017.

[134] A. Korba, A. Salim, M. Arbel, G. Luise, and A. Gretton, *A non-asymptotic analysis for stein variational gradient descent*, *Advances in Neural Information Processing Systems* **33** (2020).

[135] G. O. Roberts and R. L. Tweedie, *Exponential convergence of langevin distributions and their discrete approximations*, *Bernoulli* (1996) 341–363.

[136] G. Detommaso, T. Cui, Y. Marzouk, R. Scheichl, and A. Spantini, *A stein variational newton method*, *arXiv:1806.03085* (2018).

[137] B. C. Goodwin, *Oscillatory behavior in enzymatic control processes*, *Advances in enzyme regulation* **3** (1965) 425–437.

[138] B. Calderhead and M. Girolami, *Estimating bayes factors via thermodynamic integration and population mcmc*, *Computational Statistics & Data Analysis* **53** (2009), no. 12 4028–4045.

[139] T. Cui, K. J. Law, and Y. M. Marzouk, *Dimension-independent likelihood-informed mcmc*, *Journal of Computational Physics* **304** (2016) 109–137.

[140] P. Izmailov, W. J. Maddox, P. Kirichenko, T. Garipov, D. Vetrov, and A. G. Wilson, *Subspace inference for bayesian deep learning*, in *Uncertainty in Artificial Intelligence*, pp. 1169–1179, PMLR, 2020.

[141] F. Wenzel, K. Roth, B. S. Veeling, J. Świkatkowski, L. Tran, S. Mandt, J. Snoek, T. Salimans, R. Jenatton, and S. Nowozin, *How good is the bayes posterior in deep neural networks really?*, *arXiv preprint arXiv:2002.02405* (2020).

[142] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, *Averaging weights leads to wider optima and better generalization*, *arXiv preprint arXiv:1803.05407* (2018).

[143] J. Zhuo, C. Liu, J. Shi, J. Zhu, N. Chen, and B. Zhang, *Message passing stein variational gradient descent*, in *International Conference on Machine Learning*, pp. 6013–6022, 2018.

[144] D. Wang, Z. Tang, C. Bajaj, and Q. Liu, *Stein variational gradient descent with matrix-valued kernels*, *Advances in neural information processing systems* **32** (2019) 7834.

[145] P. Chen, K. Wu, J. Chen, T. O'Leary-Roseberry, and O. Ghattas, *Projected stein variational newton: A fast and scalable bayesian inference method in high dimensions*, *arXiv preprint arXiv:1901.08659* (2019).

[146] L. Liang, J. Xu, L. Deng, M. Yan, X. Hu, Z. Zhang, G. Li, and Y. Xie, *Fast search of the optimal contraction sequence in tensor networks*, *IEEE Journal of Selected Topics in Signal Processing* (2021).

[147] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, *Mnasnet: Platform-aware neural architecture search for mobile*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

[148] C. Li and Z. Sun, *Evolutionary topology search for tensor network decomposition*, in *International Conference on Machine Learning*, pp. 5947–5957, PMLR, 2020.

[149] C. Gong, J. Peng, and Q. Liu, *Quantile stein variational gradient descent for batch bayesian optimization*, in *International Conference on Machine Learning*, pp. 2347–2356, PMLR, 2019.

[150] Y. Du and I. Mordatch, *Implicit generation and generalization in energy-based models*, *arXiv preprint arXiv:1903.08689* (2019).

[151] K. Neklyudov, M. Welling, E. Egorov, and D. Vetrov, *Involutive mcmc: a unifying framework*, in *International Conference on Machine Learning*, pp. 7273–7282, PMLR, 2020.