DLR-IB-RM-OP-2021-181

Progressive Bayesian Neural Networks

Masterarbeit

Dominik Schnaus

DLR Deutsches Zentrum für Luft- und Raumfahrt

Institut für Robotik und Mechatronik	BJ.: 2021
	IB.Nr.: DLR-IB-RM-OP-2021-181

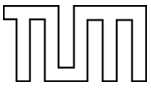
MASTERARBEIT

PROGRESSIVE BAYESIAN NEURAL NETWORKS

Freigabe:	Der Bearbeiter:	Unterschriften
	Dominik Schnaus	
	Betreuer:	
	Jongseok Lee & Dr. Rudolph Triebel	
	Der Institutsdirektor	
	Prof. Alin Albu-Schäffer	

Dieser Bericht enthält 86 Seiten, 19 Abbildungen und 9 Tabellen

Ort: Oberpfaffenhofen	Datum:	Bearbeiter:	Zeichen:



Technische Universität München

Department of Mathematics

Master's Thesis

Progressive Bayesian Neural Networks

Dominik Schnaus

Supervisor: Prof. Dr. Daniel Cremers

Advisors: Jongseok Lee & PD Dr. habil. Rudolph Triebel

Submission Date: November 01, 2021

I assure the single handed composition of this master's thesis only supported by declared resources.

Garching, November 01, 2021

Acknowledgments

Foremost, I would like to thank Dr. Rudolph Triebel for offering me this work and for the great comments on the K-FOC method, and Jongseok Lee for the insightful discussions, for always being available for questions and a large amount of constructive feedback to improve the thesis. Furthermore, I wish to show my gratitude to Prof. Daniel Cremers for supervising my thesis. In addition, I appreciate the German Aerospace Center and especially the Robotics and Mechatronics Center for access to the GPU cluster and infrastructure. Finally, I am grateful for the support of my family, Kim Le and Erik Steiger in proofreading.

Abstract

Uncertainty estimates are crucial in many deep learning problems, *e.g.* for active learning or safety-critical applications. While Bayesian deep learning provides a framework to generate uncertainty estimates for deep learning models, it requires a well-specified prior which is in general unknown.

This work aims to use large-scale datasets to learn an informative prior over the parameters of a neural network which can then be used in subsequent tasks to create better uncertainty estimations and tighter generalization bounds. The model uses scalable Laplace approximations to enable working with large-scale networks and datasets with little computational overhead compared to standard deep learning. Altogether, this transforms the problem of defining high-dimensional prior distributions with complex interactions between different weights to finding related datasets.

To improve the generalization bounds for Laplace approximation, a novel method to scale the curvature using PAC-Bayesian bounds is proposed. For this, an approximate upper bound of the training error is derived for Laplace approximation that is optimized with respect to the curvature scales.

Empirically, the learned prior needs less temperature scaling than isotropic Gaussian priors and produces similarly accurate predictions and uncertainty estimations. Moreover, non-vacuous generalization bounds are obtained for a LeNet-5 architecture on the NotMNIST dataset. In particular, the curvature scaling improves the bounds by up to 23 percent points while the empirically learned prior tightens the bound compared to isotropic Gaussian priors by an average of nine percent points, resulting in an upper bound of the generalization error of 65% on the NotMNIST dataset.

Additionally, we introduce Progressive Bayesian Neural Networks (PBNN) that combine the learned prior with progressive neural networks to learn sequentially incoming tasks without catastrophic forgetting. Using an empirically learned prior on the ImageNet dataset, PBNN improve the accuracy and uncertainty on a large-scale robotics dataset compared to progressive neural networks and their variation with MC dropout.

Moreover, we present a more accurate Kronecker-factorization of the Fisher Information Matrix (FIM) as an alternative to the widely adopted Kronecker-Factored Approximate Curvature (K-FAC). For this, we transform the optimal Kronecker-factored approximation of the FIM into a best rank-one approximation problem and solve this problem with a novel scalable version of the well-known power (iteration) method. In a proof-of-concept experiment, we show that the proposed algorithm can achieve more accurate estimates of the true FIM when compared to the K-FAC method.

Zusammenfassung

Die Unsicherheit von Deep-Learning Modellen ist von zentraler Bedeutung wie beispielsweise beim Aktiven Lernen oder bei sicherheitskritischen Anwendungen. Bayes'sches Deep Learning bietet zwar einen Rahmen für die Erstellung von Unsicherheitsschätzungen für Deep-Learning-Modelle, benötigt dafür aber eine gut definierte A-priori-Wahrscheinlichkeitsverteilung, die im Allgemeinen unbekannt ist.

Diese Arbeit hat das Ziel, große Datensätze zu verwenden, um eine informative A-priori-Wahrscheinlichkeitsverteilung über die Parameter des neuronalen Netzwerks zu erlernen, die für weitere Aufgaben verwendet werden kann und die Unsicherheitsabschätzung und Generalisierungsschranken darauf verbessern soll. Das Modell nutzt skalierbare Laplace Approximationen, um mit großen Netzwerken und Datensätzen zu arbeiten und benötigt dabei nur geringfügig mehr Rechenaufwand als normales Deep Learning. Insgesamt wird dadurch das Problem, eine hoch-dimensionale A-priori-Verteilung mit komplexen Zusammenhängen zu bestimmen, dahin verlagert, einen ähnlichen Datensatz zu finden.

Für die Verbesserung der Generalisierungsschranken für die Laplace Approximation wird eine innovative Methode vorgeschlagen, die die Krümmung mit Hilfe von PAC-Bayes'schen Schranken anpasst. Dafür wird eine approximative obere Schranke des Fehlers auf den Trainingsdaten für die Laplace Approximation hergeleitet, die bezüglich der Krümmungsskalierung optimiert wird.

In der Praxis braucht die gelernte A-priori-Wahrscheinlichkeitsverteilung weniger Temperature-scaling als isotrope Normalverteilungen und liefert ähnlich genaue Vorhersagen und Unsicherheitsabschätzungen. Darüber hinaus werden für eine LeNet-5-Architektur auf dem NotMNIST-Datensatz nicht-triviale Generalisierungsschranken erzielt. Insbesondere verbessert die Krümmungsskalierung die Generalisierungsschranke um 23 Prozentpunkte während die gelernte A-priori-Verteilung durchschnittlich mit einer Verbesserung von neun Prozentpunkten im Vergleich zu isotropischen Normalverteilungen einhergeht. Insgesamt führt die Kombination beider Methoden zu einer Schranke des Generalisierungsfehlers von 65% auf dem NotMNIST-Datensatz.

Zusätzlich stellen wir Progressive Bayes'sche Neuronale Netzwerke (PBNN) vor, die die gelernte A-priori-Verteilung mit Progressive Neural Networks kombinieren, um sequentiell eingehende Aufgaben ohne katastrophales Vergessen zu lernen. Mit einer auf ImageNet gelernten A-Priori-Verteilung verbessern PBNN die Genauigkeit und Unsicherheit im Vergleich zu Progressive Neural Networks und deren Erweiterung mit MC-Dropout.

Darüber hinaus stellen wir eine genauere Kronecker-Faktorisierung der Fisher-Informationsmatrix (FIM) als Alternative zur weit verbreiteten Kronecker-Factored Approximate Curvature (K-FAC) vor. Dafür wandeln wir die optimale Approximation der FIM als Kronecker-Faktorisierung in ein optimales Rang-Eins-Problem um und lösen dieses Problem mit einer skalierbaren Version der bekannten Power-(Iterations-)Methode. In einem Konzeptversuch zeigen wir, dass der vorgeschlagene Algorithmus im Vergleich zur K-FAC-Methode genauere Näherungen der FIM erreicht.

Contents

List of Figures

List of Tables

Notation

Sets

$\mathbb{N} = \{1, 2, \dots\}$, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$	natural numbers
$\mathbb{R}$	real numbers
$[n] = \{1, \dots, n\}$, $[n]_0 = [n] \cup \{0\}$	set of numbers up to $n \in \mathbb{N}$
$\lvert M \rvert$	the cardinality of the set M

Linear Algebra

$\mathbf{I} = \mathbf{I}^{-1}$	unit matrix
$\mathbb{1}$	all-ones vector
$\odot$	Hadamard/element-wise product
$\otimes$	Kronecker product
$\mathrm{tr}(\mathbf{A}) = \sum_{i=1}^{m} \mathbf{A}_{i,i}$	trace of $\mathbf{A} \in \mathbb{R}^{m \times m}$
$\mathrm{vec}(\mathbf{A}) = (\mathbf{A}_{1,1}, \mathbf{A}_{1,2}, \dots, \mathbf{A}_{m,n})^T$	(row-)vectorization of $\mathbf{A} \in \mathbb{R}^{m \times n}$
$\mathrm{diag}((\mathbf{A}^i)_{i \in [k]}) = \begin{pmatrix} \mathbf{A}^1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{A}^k \end{pmatrix}$	block-diagonal matrix with $\mathbf{A}^i \in \mathbb{R}^{m_i \times n_i}$
$\mathrm{diag}(\mathbf{A}) = (\mathbf{A}_{1,1}, \mathbf{A}_{2,2}, \dots, \mathbf{A}_{m,m})^T$	diagonal of the matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$
$\Lambda(\mathbf{A})$	set of eigenvalues for the matrix $\mathbf{A}$
$\mathbf{A}_{:,i} \in \mathbb{R}^m$	i-th column of $\mathbf{A} \in \mathbb{R}^{m \times n}$
$\mathbb{1}[s] = \begin{cases} 1 & \text{if } s \\ 0 & \text{else} \end{cases}$	0-1 function for statement s

Probability Theory

π and $\pi(\cdot) = p(\cdot)$	prior distribution and p.d.f.
ρ and $\rho(\cdot) = p(\cdot \vert \mathcal{D})$	posterior distribution and p.d.f.
$\mathcal{N}(\mu, \Sigma)$ and $\mathcal{N}(\cdot \vert \mu, \Sigma)$	normal distribution and p.d.f.
$\mathcal{MN}(\mu, \Sigma)$ and $\mathcal{MN}(\cdot \vert \mu, \Sigma)$	matrix normal distribution and p.d.f.
$p \ll q$	p is absolutely continuous w.r.t. q
$\mathbb{KL}(p \vert\vert q) = \begin{cases} \int \ln\left(\frac{dp}{dq}\right) dp & \text{if } p \ll q \\ \infty & \text{else} \end{cases}$	KL-divergence for distributions p and q

Neural Networks

$l \in [L]$	layer index
σ_l	activation function
ϕ_l	parameterized function
$\mathbf{s}_l \in \Omega_l$	pre-activation
$\mathbf{a}_l \in \Omega_l$	activation
$\mathcal{X} = \Omega_0$	input space
$\mathcal{Y}$	target space
$\mathbf{x} = \mathbf{a}_0 \in \mathcal{X}$	input
$\mathbf{y} \in \mathcal{Y}$	target
$\mathbf{W}^l$	weight
$\mathcal{D} = ((\mathbf{x}_i, \mathbf{y}_i))_{i=1}^N$	dataset
$(\mathbf{x}, \mathbf{y}) \sim P$	data distribution
$\boldsymbol{\theta}_l = \mathrm{vec}(\mathbf{W}^l) \in \mathbb{R}^{n_l}$	parameter vector for layer l
$\boldsymbol{\theta} = \mathrm{vec}((\boldsymbol{\theta}_l)_{l \in [L]}) \in \mathbb{R}^n$	full parameter vector
$f_{\boldsymbol{\theta}}$	neural network with weight vector $\boldsymbol{\theta}$
$p(\cdot \vert \mathbf{x}, f_{\boldsymbol{\theta}})$	distribution over $\mathcal{Y}$ defined by $f_{\boldsymbol{\theta}}$
$\gamma > 0$	weight decay

Fully-Connected Layer

$\Omega_l = \mathbb{R}^{d_l}$	feature space
$\mathbf{W}^l \in \mathbb{R}^{d_l \times (d_{l-1}+1)}$	weight matrix
$n_l = d_l(d_{l-1} + 1)$	dimension of $\boldsymbol{\theta}^l$

Convolutional Layer

$\Omega_l = \mathbb{R}^{c_l \times h_l \times w_l}$	feature space
$\mathcal{T}^l = [h_l] \times [w_l]$	feature spatial positions
$\Delta^l = [h_l^\Delta] \times [w_l^\Delta]$	kernel spatial positions
$\mathbf{W}^l \in \mathbb{R}^{c_l \times (c_{l-1} \vert \Delta^l \vert + 1)}$	weight matrix
$n_l = c_l(c_{l-1} \vert \Delta^l \vert + 1)$	dimension of $\boldsymbol{\theta}^l$

PAC-Bayesian Bounds

$\mathcal{G}$	hypothesis set
$\mathcal{M}$	set of probability distributions over $\mathcal{G}$
$l : \mathcal{G} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$	loss function
$\mathrm{er} : \mathcal{G} \times \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$	0-1-loss / error loss
$\mathcal{L}_P^l(g) = \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim P}\left[l(g, \mathbf{x}, \mathbf{y})\right]$	risk for $g \in \mathcal{G}$
$\hat{\mathcal{L}}_{\mathcal{D}}^l(g) = \frac{1}{N} \sum_{i=1}^N l(g, \mathbf{x}_i, \mathbf{y}_i)$	empirical risk for $g \in \mathcal{G}$

Laplace Approximation

$\mathbf{H} \in \mathbb{R}^{n \times n}$	Hessian matrix
$\mathbf{F} \in \mathbb{R}^{n \times n}$	Fisher Information Matrix (FIM)
$\mathbf{E}_{(\mathbf{x},\_)\sim P}\mathbf{E}_{\mathbf{y}\sim p(\cdot\|\mathbf{x},f_{\boldsymbol{\theta}})}$	expectation of model distribution
$\mathcal{D}\cdot = \frac{d \ln p(\mathbf{y}\|\mathbf{x},f_{\boldsymbol{\theta}})}{d\cdot}$	derivative of the log-likelihood
$\mathbf{F}_l \in \mathbb{R}^{n_l \times n_l}$	diagonal block of $\mathbf{F}$ corresponding to layer l
$\mathbf{L}_l \otimes \mathbf{R}_l \approx \mathbf{F}_l$	Kronecker-factored approximation (K-FAC or K-FOC)

Continual Learning

$\mathfrak{T}_t = (P_t, \mathcal{D}_t)$	task t
$\mathcal{D}_t = \left((\mathbf{x}_i^{(t)}, \mathbf{y}_i^{(t)})\right)_{i=1}^{N_t}$	dataset
$(\mathbf{x}_i^{(t)}, \mathbf{y}_i^{(t)}) \sim P_t$	data distribution
$\mathcal{X} = \mathcal{X}_t = \mathcal{X}_{t'}$	input space
$\mathcal{Y}_t$	target space

Progressive (Bayesian) Neural Networks

$\mathfrak{L} \subset [L]$	lateral connections
α_l, β_l	curvature scales
τ	temperature scaling (same for all priors and posteriors)
$\tilde{\mathbf{F}}_l$	precision matrix
superscript $\cdot^{(t',t)}$	from column t' to column t
superscript $\cdot^{(t)}$	incoming to column t
superscript $\cdot^{(\leq t)}$	all columns up to t
superscript $\cdot^{(0)}$	prior

Abbreviations

MLE	Maximum Likelihood Estimation	EWC	Elastic Weight Consolidation
MAP	Maximum-A-Posteriori	OSLA	Online Structured Laplace Approximation
ReLU	Rectified Linear Unit	PNN	Progressive Neural Networks
BNN	Bayesian Neural Network	PBNN	Progressive Bayesian Neural Networks
KL	Kullback–Leibler	OOD	Out-Of-Distribution
MC	Monte Carlo	ID	In-Distribution
MCMC	Markov Chain Monte Carlo	ECE	Expected Calibration Error
ELBO	Evidence Lower BOund	*e.g.*	for example (exempli gratia)
FIM	Fisher Information Matrix	*i.e.*	that is (id est)
K-FAC	Kronecker-Factored Approximate Curvature	w.r.t.	with respect to
K-FOC	Kronecker-Factored Optimal Curvature	p.d.f.	probability density function
PAC	Probably Approximately Correct		

1. Introduction

1.1. Motivation and Problem Statement

In recent years, deep learning methods have excelled in numerous fields, including computer vision [85] and natural language processing [12, 22], even reaching human-like performance in terms of accuracy [17, 42]. With ongoing improvements, deep neural networks are more and more included in safety-critical applications, *e.g.* for autonomous driving [37, 48] or the medical domain [40, 97, 104]. This strengthens the importance of reasonable uncertainty estimates to observe and react to unknown situations and to detect out-of-distribution samples. Additionally, practical uncertainty estimates play an important role in active learning, robustness, and the detection of new classes [28, 32].

Bayesian neural networks [6] are one way to improve uncertainty estimations in deep learning [32]. They use Bayes' rule to include probabilistic inference in deep learning by learning distributions over the network parameters. For this, an open problem is how the high-dimensional prior distribution over the parameters should be specified in advance [112]. A well-specified prior should capture the complex interactions between the weights introduced by the neural network and reasonably restrict the search space during learning. Recent work suggests that the choice of the prior plays a significant role to make Bayesian neural networks practically relevant [30, 112, 113]. Existing methods mainly use uninformative priors like isotropic Gaussians or hierarchical distributions [30]. Functional priors avoid the problem of specifying the prior in the weight space by introducing priors over the functions defined by the neural network rather than over the weights. Still, most of these methods are not scalable to large networks and datasets [53, 102]. In contrast to the standard Bayesian theory where the prior is chosen before any data is seen, empirical Bayes' methods use data to learn a prior distribution [95]. This data can either be chosen to be the dataset of interest [57] or a dataset of a similar domain [56]. For standard deep learning, a common method is to use a pre-trained neural network on a different dataset and fine-tune either some layers or the full network on the target task [12, 104]. This employs that many low-level features can be shared among different tasks [105, 119]. In this thesis, we utilize this idea to learn the prior.

Given a prior distribution, it is usually infeasible to compute the posterior over the parameters exactly [32]. Besides sampling and variational methods, Laplace approximation tries to find a feasible distribution that is close to the true posterior. For this, the log-posterior is approximated at a mode with its second-order Taylor polynomial to find a normal distribution that is close to the posterior around the mode [67, 92]. Still, for

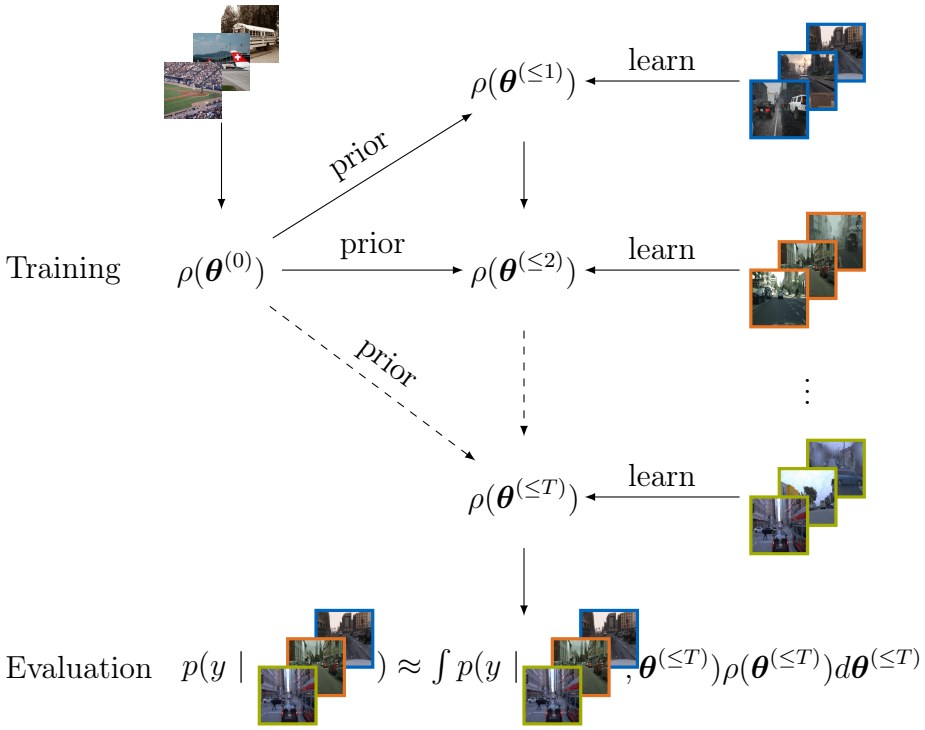


Figure 1.1.: At first, a prior is learned from a large-scale dataset (ImageNet [20]). This prior is then used sequentially for each other task. The first task (blue) corresponds to simulation data [90], the second task (orange) corresponds to an autonomous driving dataset from Germany [18] and both datasets should improve the performance on the final autonomous driving dataset in the USA [100] (green). Altogether, the algorithm should have the ability to do Bayesian model averaging by approximating the posterior over the parameters. Finally, the model is evaluated on all tasks except ImageNet.

modern neural networks, it is usually not possible to compute the full Hessian. As a surrogate either diagonal or Kronecker-factored approximations of the Fisher Information Matrix (FIM) are used. In practice, the Kronecker-Factored Approximate Curvature [38, 71] (K-FAC) is a widely used method that approximates the FIM with a block diagonal matrix where each block is Kronecker-factored and corresponds to one layer of the neural network. Hence, the K-FAC provides a good balance between computational tractability and the exactness of the approximation [2]. However, the K-FAC also makes significant assumptions that are often not satisfied in practice [106]. In this thesis, we introduce an alternative method that approximates the FIM more accurately. Using these approximations of the FIM, computing the Laplace approximation only needs little computational overhead compared to standard maximum-a-posteriori (MAP) training, namely one additional epoch over the training data [38, 71, 92]. In practice, the quality of the Laplace approximation is often improved by scaling the curvature of the prior or the posterior [55, 92, 93]. We propose a novel method to automatically scale both curvatures by minimizing generalization bounds for Bayesian neural networks, the so-called PAC-Bayesian bounds [13, 39].

In practice, often multiple related datasets corresponding to different tasks are available. Thus, we aim to incorporate this data in our training and use the features and distributions that were learned on these tasks. In particular, we further assume that all tasks arrive sequentially and previous tasks can not be accessed, resulting in a task-incremental continual learning problem [47]. To use the previous features, we build on top of progressive neural networks [96, 111] (PNN). This is a method that uses one network copy, the so-called column, for each task. Moreover, lateral connections are introduced that are weighted layers that use features from previous columns. An example of this setting, where one task is used to learn the prior and the other tasks to learn feature embeddings and distributions, can be seen in Figure 1.1 on a practical use case for autonomous driving.

1.2. Contribution

The main contribution of this thesis is a scalable method that learns informed prior distributions from data and utilizes them together with features from multiple related datasets in a continual learning setting to improve the uncertainty estimation and generalization. This includes

- a method to compute a prior distribution from a large-scale dataset which is used to improve the posterior distribution in Bayesian neural networks,
- the automatic scaling of the prior and posterior curvatures with PAC-Bayesian bounds to improve existing generalization bounds,
- the extension of this method to use features and weight distributions from multiple related tasks that come in sequential order, and
- a comparison of the Kronecker-Factored Approximate Curvature (K-FAC) with the optimal Kronecker factorization (K-FOC) of the Fisher matrix on multiple datasets.

Specifically, the empirically learned prior corresponds to the Laplace approximation of the posterior given a related large-scale dataset. This results in a multivariate normal distribution around the MAP estimated parameters which can be used to improve the posterior distribution. The experiments show that this prior needs in general less temperature scaling than isotropic Gaussian priors to achieve similar accuracy and uncertainty estimates which is an indication that this prior is better specified. Moreover, the evaluated PAC-Bayesian bounds are on average tighter on the NotMNIST[1] dataset, showing that the posterior is able to be closer to the prior with respect to the Kullback-Leibler-divergence (KL-divergence) without lacking accuracy.

Furthermore, an approximate upper bound of the expected training error for Laplace approximation is derived. Incorporating this in PAC-Bayesian bounds leads to a heuristic that is directly minimized with respect to the curvature scales. Empirically, the curvature

[1] From `http://yaroslavvb.blogspot.co.uk/2011/09/notmnist-dataset.html`

scaling in combination with the learned prior results in non-vacuous generalization bounds on the NotMNIST dataset with an upper bound of 65% on the generalization error.

PBNN extend the empirically learned prior and the curvature scaling to PNN. For this, we use previously learned weight distributions from preceding tasks as the prior on the lateral connections. This method is evaluated on a large-scale robotics dataset using a prior learned on the ImageNet dataset. This improves the accuracy and uncertainty estimates compared to PNN.

For the K-FOC method, we transform the problem of finding the closest Kronecker-factored matrix with respect to the FIM into a best rank-one problem. Furthermore, we derive a scalable variant of the power-iteration method that solves this problem with similar computational complexity as the K-FAC.

1.3. Related Work

The closest related work for our empirical prior learning is the Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting [93]. In a continual learning setting, this method updates the prior distribution for a new task with the posterior of the last task. Moreover, Laplace approximation is used together with K-FAC to estimate the posterior in each task. Hence, their prior given the first task is similar to our learned prior. Nevertheless, this prior is not used in combination with Bayesian neural networks but to regularize the training. In a different work, Ritter *et al.* [92] use Laplace approximation with K-FAC as a Bayesian neural network but only specify isotropic Gaussian priors. Moreover, in both works, the curvature scales are found manually. The resulting empirical Gaussian prior around the MAP parameters in our prior is also related to the prior introduced by Krishnan *et al.* [56]. They also learn the prior mean over a related dataset with deterministic pre-training but use a hierarchical model on the variance. In addition, they use variational inference to approximate the posterior instead of Laplace approximation. Lee *et al.* [63] also use a pre-trained neural network and Laplace approximation together with K-FAC but generate sampling-free uncertainty estimates with a mixture of Gaussian processes.

Similar to our approach of scaling the curvature using PAC-Bayesian bounds, there were previous methods that directly optimize PAC-Bayesian bounds to obtain tighter and non-vacuous generalization bounds. Dziugaite & Roy [25] use a PAC-Bayesian bound to find the parameters and the diagonal covariance matrix of a Gaussian posterior distribution. Still, this method uses an isotropic Gaussian prior and does not use Laplace approximation. More related to our bound using Laplace approximation is the approach by Pitas [86], where PAC-Bayesian bounds are combined with Taylor approximation to jointly find a Gaussian prior and posterior. In contrast to our approach, the $\mathcal{L}_2$-loss is used instead of the error or the negative log-likelihood. Additionally, we optimize the curvature scales of the posterior to obtain a prior that is independent of the training data.

PBNN aim to leverage multiple datasets to improve performance on the following tasks. This is closely related to transfer learning and continual learning. A common method for transfer learning is deterministic pre-training [105, 119] which learns a network on a dataset and uses these weights as initialization or a low-level feature extractor for another dataset. Anderson *et al.* [1] used a similar architecture as PNN [96] for transfer learning to improve the performance for small amounts of data. This work is extended by Wang *et al.* [111] for the robotics domain. Moreover, PNN are also a common method to improve the performance in the continual learning setting given multiple tasks. We combine the structure of PNN with the empirically learned prior and the curvature scaling for PBNN.

For K-FOC, the recent work of Kao *et al.* [52] pioneers the K-FAC method as an optimization problem over the sums of Kronecker products in the context of continual learning for recurrent neural networks. However, Kao *et al.* [52] rely on solvers that are cubic in cost (for example, QR and singular value decomposition) which can be prohibitively too expensive for the parameter space of deep neural networks, especially for convolutional layers. Instead, we introduce a scalable version of the power method to solve the optimization problem. In this sense, our method presents for the first time to our knowledge, a scalable algorithm that imposes the K-FAC method as an optimization problem.

1.4. Thesis Structure

The thesis is structured as below. The background of PBNN is provided in Chapter 2. Chapter 3 first explains how informed priors can be learned from data in the transfer learning setting and how PAC-Bayesian bounds can be used to improve the generalization bounds for Laplace approximation in Section 3.1. The extension to multiple tasks in the continual learning setting is presented in Section 3.2. The empirical experiments are shown in Chapter 4 and the thesis is concluded in Chapter 5. Moreover, the details for K-FOC and a comparison with K-FAC can be found in Appendix A.1.

2. Background

PBNN combine Bayesian neural networks with continuous learning. In particular, they build on Laplace approximation, PAC-Bayesian bounds, and progressive neural networks. In this chapter, we first give an overview of the concept of Bayesian neural networks based on standard deep learning with an emphasis on Laplace approximation and commonly used priors. Moreover, we discuss the main idea of the PAC-Bayesian theory. We then conclude the chapter with continuous learning, focusing on progressive neural networks.

2.1. Deep Learning

Within the last decade, deep learning techniques achieved or surpassed human-like performance in a variety of fields like computer vision [42, 85], natural language processing [12, 22], and medicine [40, 97, 104]. Deep learning is a machine learning technique that mainly builds on top of *artificial neural networks*, *i.e.* parameterized functions that are compositions of multiple linear and non-linear layers [35].

Each layer $l \in [L] := \{1, \dots, L\}$ of an artificial neural network $f_{\boldsymbol{\theta}}$ with $L \in \mathbb{N}$ layers consists of a parameterized function $\phi_l : \Omega_{l-1} \to \Omega_l$ and a non-linear activation function $\sigma_l : \Omega_l \to \Omega_l$. Let $\mathbf{x} \in \Omega_0 =: \mathcal{X}$ be an input for a network $f_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$. Then the *pre-activations* $\mathbf{s}_l$ and *activations* $\mathbf{a}_l$ are defined recursively with $\mathbf{a}_0 = \mathbf{x}$, $\mathbf{s}_l = \phi_l(\mathbf{a}_{l-1})$ and $\mathbf{a}_l = \sigma_l(\mathbf{s}_l)$. Using these intermediate results, the output of the neural network is given by the last activation, namely $f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{a}_L$.

From a probabilistic perspective, the output of a neural network can often be seen as the parameters of a probability distribution $p(\mathbf{y}|\mathbf{x}, f_{\boldsymbol{\theta}})$ over the target space $\mathcal{Y}$ [71]. The most prominent example is a categorical distribution that is induced by the outputs of a softmax function in the last layer:

$$\sigma_L : \mathbb{R}^{d_L} \to \mathbb{R}^{d_L},\ \sigma_L(\mathbf{x})_i = \frac{\exp \mathbf{x}_i}{\sum_{j=1}^{d_L} \exp \mathbf{x}_j}.$$

Layers. In computer vision tasks, the parameterized function ϕ_l is often a fully-connected or a convolutional layer [110]. In this paragraph, we consider one specific layer $l \in [L]$. For this, we drop the layer index l when it is not needed and define the corresponding pre-activation as $\mathbf{s} := \mathbf{s}_l$ and the activation from the previous layer as $\mathbf{a} := \mathbf{a}_{l-1}$.

Fully-Connected Layer. Mathematically, a *fully-connected layer* [35] is a matrix product on a vectorized input, *i.e.*

$$\Omega_{l-1} = \mathbb{R}^{d_{l-1}}, \quad \Omega_l = \mathbb{R}^{d_l}$$
$$\phi(\mathbf{a}) = \bar{\mathbf{W}}\mathbf{a} + \mathbf{b} = \begin{pmatrix} \bar{\mathbf{W}} & \mathbf{b} \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ 1 \end{pmatrix} =: \mathbf{W}\bar{\mathbf{a}},$$

where $\bar{\mathbf{W}} \in \mathbb{R}^{d_l \times d_{l-1}}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^{d_l}$ is the bias and $\mathbf{W} \in \mathbb{R}^{d_l \times (d_{l-1}+1)}$ is the concatenation of both.

Convolutional Layer. In contrast, *convolutional layers* are usually not applied on vectors but are defined on tensors [35]. For image data, the input and output of a convolutional layer is a three-tensor, *i.e.* $\Omega_{l-1} = \mathbb{R}^{c_{l-1} \times h_{l-1} \times w_{l-1}}$ and $\Omega_l = \mathbb{R}^{c_l \times h_l \times w_l}$. The weight is a four-tensor $\bar{\mathbf{W}} \in \mathbb{R}^{c_l \times c_{l-1} \times h_l^\Delta \times w_l^\Delta}$ and the bias is a vector $\mathbf{b} \in \mathbb{R}^{c_l}$. Such a layer transforms the input $\mathbf{a} \in \Omega_{l-1}$ by sliding the weight matrix over the image and computing a dot product of the weight with the corresponding input elements at each spatial position. In this thesis, the notation of Martens & Grosse [71] with multi-indices for the spatial positions is used. This means that the height and width indices of the output tensor are summarized with one multi-index

$$\mathcal{T} = [h_l] \times [w_l]$$

and the spatial positions of the weight tensor are defined as

$$\Delta = [h_l^\Delta] \times [w_l^\Delta].$$

For a stride $\mathbf{r} \in \mathbb{N}^2$, padding $\mathbf{p} \in \mathbb{N}^2$ and the index function $\zeta(\mathbf{t}, \delta) = (\mathbf{t} - \mathbb{1}) \odot \mathbf{r} - \mathbf{p} + \delta$ a convolutional layer can be defined as

$$\phi(\mathbf{a})_{k,\mathbf{t}} = \mathbf{b}_k + \sum_{k'=1}^{c_{l-1}} \sum_{\delta \in \Delta} \bar{\mathbf{W}}_{k,k',\delta} \mathbf{a}_{k',\zeta(\mathbf{t},\delta)}.$$

Here, we define $\mathbf{a}_{k',\mathbf{t}'} = 0$ if $\mathbf{t}' \notin [h_{l-1}] \times [w_{l-1}]$ to zero-pad the tensor outside its definition. Moreover, $\odot$ is the Hadamard product and $\mathbb{1}$ is the all-ones vector.

A convolutional layer is a special case of a fully-connected layer where some weights are shared among different spatial positions [71]. To observe this, define the expansion $\hat{\mathbf{a}}$ of $\mathbf{a}$ as the $c_{l-1}|\Delta| \times |\mathcal{T}|$-sized matrix where column $\mathbf{t}$ is given by the vector

$$\hat{\mathbf{a}}_{:,\mathbf{t}} = \mathrm{vec}\big((\mathbf{a}_{k',\zeta(\mathbf{t},\delta)})_{k' \in [c_{l-1}], \delta \in \Delta}\big),$$

where $\hat{\mathbf{a}}_{:,\mathbf{t}}$ denotes the $\mathbf{t}$-th column of the matrix $\hat{\mathbf{a}}$. By viewing the weight tensor as a matrix in $\mathbb{R}^{c_l \times c_{l-1}|\Delta|}$ one can compute the convolutional layer as a matrix-matrix product

$$\phi(\mathbf{a}) = \mathbf{b} + \bar{\mathbf{W}}\hat{\mathbf{a}} = \begin{pmatrix} \bar{\mathbf{W}} & \mathbf{b} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{a}} \\ \mathbb{1}^T \end{pmatrix} =: \mathbf{W}\bar{\mathbf{a}}.$$

Pooling Layer. Additional to convolutional and fully connected layers, computer vision tasks often include *pooling layers* [35] that reduce the spatial size by a non-parametric function, *e.g.* the maximum or the average, applied on a local neighborhood.

Optimal Parameters. The parameters $\boldsymbol{\theta}$ of a neural network given independent training samples $\mathcal{D} = ((\mathbf{x}_i, \mathbf{y}_i))_{i=1}^N \sim P^N$ from an unknown distribution P over $\mathcal{X} \times \mathcal{Y}$ can be obtained by maximum likelihood or maximum-a-posteriori estimation [35].

Maximum likelihood estimation (MLE) finds the parameters that maximize the probability of the training samples given the specific network architecture, *i.e.*

$$\boldsymbol{\theta}^{MLE} \in \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x}_i, f_{\boldsymbol{\theta}}).$$

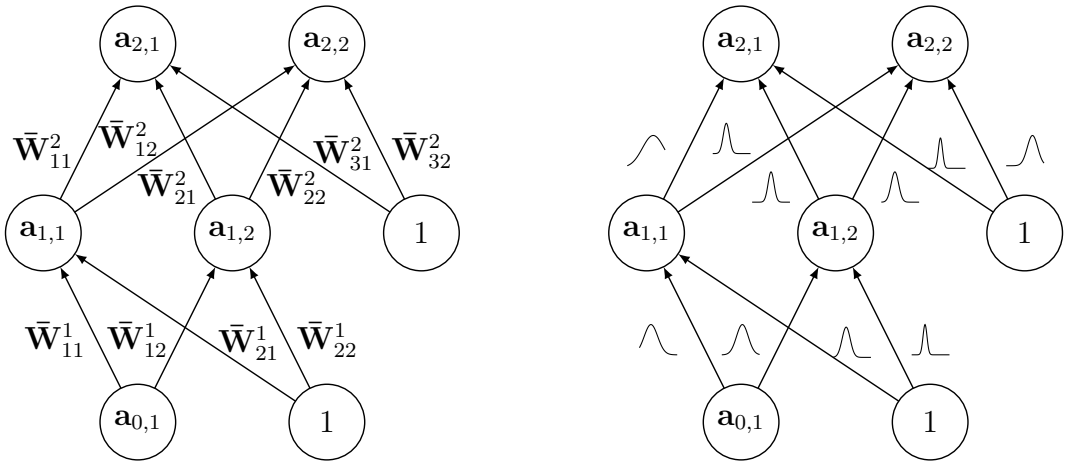
However, MLE is prone to overfit training samples and hence often does not generalize well to unseen data [51]. To reduce overfitting, *maximum-a-posteriori* (MAP) estimation introduces the possibility to include a prior distribution over the parameters. Given a probability density function $\pi(\boldsymbol{\theta})$ of a prior distribution over the parameter vector of the neural network, the MAP estimate is the parameter with the highest posterior probability

$$\begin{aligned}\boldsymbol{\theta}^{MAP} &\in \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}) \\ &= \arg\max_{\boldsymbol{\theta}} \pi(\boldsymbol{\theta}) \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x}_i, f_{\boldsymbol{\theta}}),\end{aligned}$$

where Bayes' rule is used to formulate the posterior in terms of the prior and the likelihood.

To improve the optimization process to find the optimal parameters, a lot of improvements have been proposed to the basic network architecture [43, 50, 103, 118]. In this thesis, especially residual networks [43] (ResNets) and batch normalization [50] are used. ResNets use skip connections, which means that activations from earlier layers are added to the pre-activations of successive layers, *i.e.* $\mathbf{s}_l = \phi_l(\mathbf{a}_{l-1}) + \mathbf{a}_j$ for $j < l$. Batch normalization uses the current batch of samples to compute the mean and standard deviation of pre-activations. With these statistics the pre-activation is normalized. Additionally, batch normalization introduces two learnable scalars to scale and shift the pre-activations.

Deterministic Pre-Training. A widespread method to improve the initialization and the sample complexity in neural network training is *deterministic pre-training* [105, 119]. This means that a model trained on another dataset is used for initialization. Moreover, the early layers are often fixed and only the last layers are optimized. Deterministic pre-training achieves great success in a lot of areas including medical image processing [104], natural language processing [12], and computer vision [105]. PBNN use that low-level feature extractors defined by the early layers of neural networks can be shared among tasks and extend this idea to distributions of parameters.



(a) Fully-connected neural network. (b) Fully-connected Bayesian neural network.

Figure 2.1.: While usual neural networks learn specific weights for each layer, Bayesian neural networks learn distributions over these weights.

2.2. Bayesian Neural Networks

In standard deep learning, usually, one set of weights is learned given either by MLE or MAP. From a statistical point of view, point estimates provide relatively little information about the posterior distribution. The lack of expressiveness is often reflected by a bad generalization and overconfidence on out-of-distribution samples [51]. *Bayesian neural networks* address this issue by estimating the posterior distribution over the weights given the prior π

$$\rho(\boldsymbol{\theta}) := p(\boldsymbol{\theta}|\mathcal{D}) \propto \pi(\boldsymbol{\theta}) \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x}_i, f_{\boldsymbol{\theta}}).$$

With this, for each parameter, not a single value but a distribution over multiple parameter settings is learned as shown in Figure 2.1. This allows describing the uncertainty of the model by the posterior distribution over the corresponding weights [32]. During inference, one can utilize marginalization to compute the predictive probability of a new sample $(\mathbf{x}^*, \mathbf{y}^*) \notin \mathcal{D}$ by using the posterior distribution

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) = \int p(\mathbf{y}^*|\mathbf{x}^*, f_{\boldsymbol{\theta}})\rho(\boldsymbol{\theta})d\boldsymbol{\theta}.$$

In practice, this integral is usually approximated with a Monte Carlo estimation [32] using $B_{MC} \in \mathbb{N}$ weight samples from the posterior distribution,

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) \approx \sum_{i=1}^{B_{MC}} p(\mathbf{y}^*|\mathbf{x}^*, f_{\boldsymbol{\theta}_i}), \quad \boldsymbol{\theta}_i \sim \rho.$$

For regression, other statistics like the mean and the standard deviation can also be estimated using multiple model samples. In addition to providing a nice representation

of uncertainty in Bayesian neural networks, Bayesian model averaging can also be seen as an ensemble method that combines multiple model predictions to improve the reliability of the resulting prediction. [51, 113, 114].

Posterior Approximation. Even for small-sized neural networks, it is usually intractable to compute the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ exactly. In particular, for Bayesian neural networks, it is in general not straightforward to integrate over the model parameters or to sample from the posterior. Common approximations of the posterior are sampling methods, variational inference, and Laplace approximation [32]. In the following, the three methods are introduced.

Sampling Methods. *Sampling methods* are usually based on Markov Chain Monte Carlo (MCMC) algorithms to be able to sample from the unnormalized posterior. Hence, new weights are sampled dependent only on the current ones. To sample the weights, variants of rejection sampling are commonly used. For this, an arbitrary distribution over the weight space generates samples that are rejected dependent on the unnormalized posterior probability of this and the previous sample. Therefore, subsequent samples are often correlated. Recent work particularly builds on top of Hamiltonian Monte Carlo methods that use a first-order differential equation including a momentum term to reduce the correlation between subsequent samples while keeping the acceptance rate high [79]. This is often combined with batch estimates to improve the scalability [14]. Still, current implementations have high computational and memory requirements and are usually not scalable to large-scale network architectures and datasets [51, 82].

Variational Inference. In contrast to sampling methods, *variational inference* aims to find an easier parametric probability distribution q_ϕ that is close to the true posterior. For this, the Kullback-Leibler-divergence (KL-divergence) between the variational distribution q_ϕ and the posterior is minimized. As in general the integration over the true posterior in the KL-divergence is not tractable, an equivalent optimization problem is solved by maximizing the Evidence Lower BOund (ELBO) [3, 46]. The quality of this approximation is mainly dependent on the distribution class that is chosen for q_ϕ. Usual choices are distributions from the exponential family or hierarchical distributions [8, 36, 66]. Compared to deterministic neural network training, variational inference converges slower and each step is usually noisier [51].

Laplace Approximation. Similar to variational inference, *Laplace approximation* [67] estimates the posterior with a less complicated distribution, namely a normal distribution at a mode of the posterior. This is accomplished by using the second-order Taylor polynomial of the log posterior. In contrast to the other methods, the training phase just consists of finding a MAP estimate without sampling any parameters. After finding the mode, the precision matrix of the normal distribution is obtained by computing or approximating the Hessian at this mode. Depending on the approximation, this is complexity-wise only

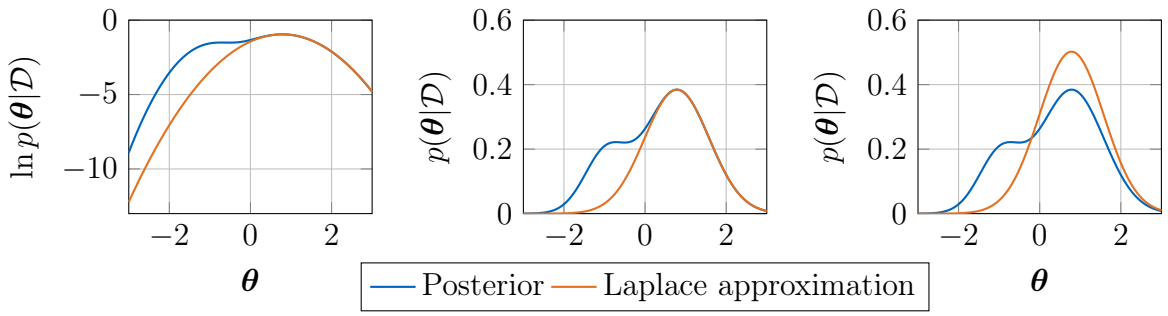


Figure 2.2.: In the left plot one can see the log-posterior and the corresponding Taylor approximation. The middle and right plot show the posterior and the unnormalized and normalized Laplace approximation, respectively.

one additional epoch compared to standard MAP neural network training [4, 10, 38, 64, 71]. Nonetheless, the main drawbacks of Laplace approximation are arguably the quality of the approximation, as Laplace approximation only captures one out of many modes of the posterior and even for this mode the estimation worsens with an increasing distance to the mode.

In this thesis, Laplace approximation is used to approximate the posterior. Therefore, the next section will discuss this topic and in particular common approximations of the Hessian for Laplace approximation.

2.3. Laplace Approximation

Laplace approximation [67] locally approximates the posterior around a mode $\hat{\boldsymbol{\theta}}$, namely the MAP estimate, by a normal distribution. For this, the second-order Taylor approximation of the log-posterior around $\hat{\boldsymbol{\theta}}$ is considered,

$$\ln p(\boldsymbol{\theta}|\mathcal{D}) \approx \ln p(\hat{\boldsymbol{\theta}}|\mathcal{D}) + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}),$$

with the Hessian of the log-posterior at the mode

$$\begin{aligned}
\mathbf{H} &= \frac{d^2}{d\boldsymbol{\theta}^2} \ln p(\boldsymbol{\theta}|\mathcal{D})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} \\
&= \frac{d^2}{d\boldsymbol{\theta}^2} \ln p(\mathcal{D}|f_{\boldsymbol{\theta}})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} + \frac{d^2}{d\boldsymbol{\theta}^2} \ln p(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} \\
&=: \mathbf{H}_{likelihood} + \mathbf{H}_{prior}.
\end{aligned}$$

This is shown in the left plot in Figure 2.2. As the Taylor approximation is around a mode, the first-order term vanishes. The unnormalized Laplace approximation (shown in the middle of Figure 2.2) can then be obtained by taking the exponential,

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\hat{\boldsymbol{\theta}}|\mathcal{D}) \exp\left(\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})\right).$$

The resulting normalized Laplace approximation is then a normal distribution at the mode with the precision matrix $\mathbf{H}$, *i.e.*

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx \mathcal{N}(\hat{\boldsymbol{\theta}}, \mathbf{H}^{-1}). \tag{2.1}$$

It is depicted in Figure 2.2 in the right plot. In practice, the uncertainty is often underestimated by Laplace approximation. Therefore, $\mathbf{H}_{likelihood}$ and $\mathbf{H}_{prior}$ are often scaled. Scaling both jointly is equivalent to the so-called temperature scaling while scaling each factor individually can also be seen as adding or removing additional identical data points to $\mathcal{D}$ or increasing the weight decay [92]. Temperature scaling is a heuristic method to improve the posterior that corresponds to a scaling of the covariance matrix for Laplace approximation [64, 112].

Fisher Information Matrix. As $\hat{\boldsymbol{\theta}}$ could be a saddle point or because of numerical instabilities, the Hessian could be indefinite in which case the normal distribution in Equation 2.1 is not well defined [10, 70]. Therefore, positive semi-definite approximations of the Hessian are used like the *Fisher information matrix* (FIM) or the *Gauss-Newton matrix*. For a likelihood of an exponential family, *e.g.* categorical and normal distributions, both approximations are the same and for piece-wise linear activation functions like ReLU [77], they moreover coincide with the Hessian [71].

Definition 2.3.1 (Fisher Information Matrix [71])**.** Let P be a distribution over $\mathcal{X} \times \mathcal{Y}$ and $p(\cdot|\mathbf{x}, f_{\boldsymbol{\theta}})$ be a conditional distribution over $\mathcal{Y}$ dependent on the parameter vector $\boldsymbol{\theta}$. Then the FIM is defined as

$$\mathbf{F} = \mathbb{E}_{(\mathbf{x},\_)\sim P}\mathbb{E}_{\mathbf{y}\sim p(\cdot|\mathbf{x},f_{\boldsymbol{\theta}})}\left[\frac{d \ln p(\mathbf{y}|\mathbf{x}, f_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}} \frac{d \ln p(\mathbf{y}|\mathbf{x}, f_{\boldsymbol{\theta}})}{d\boldsymbol{\theta}}^T\right].$$

Here, the underscore denotes that the corresponding variable is not used.

Remark. Note that the targets from the training data are not used to compute the FIM. Using the ground truth targets instead of samples from the predictive model distribution would lead to the empirical FIM.

For the sake of an easier notation, we write $\mathbb{E}$ instead of $\mathbb{E}_{(\mathbf{x},\_)\sim P}\mathbb{E}_{\mathbf{y}\sim p(\cdot|\mathbf{x},f_{\boldsymbol{\theta}})}$ and $\mathcal{D}\cdot = \frac{d \ln p(\mathbf{y}|\mathbf{x},f_{\boldsymbol{\theta}})}{d\cdot}$ for the derivative of the log-likelihood in the following. Moreover, we drop the layer index for the activations and pre-activations, *i.e.* $\bar{\mathbf{a}} = \bar{\mathbf{a}}_l$ and $\mathbf{s} = \mathbf{s}_l$.

FIM Approximations. The full FIM and even a block-diagonal approximation without correlations between different layers are usually not feasible to store or compute for modern neural networks [71]. Common approximations of the block-diagonal form are by a

diagonal or a Kronecker-factored matrix. The Kronecker-factored approximation comes from the fact that for a single sample, the FIM is the sum of Kronecker-factored matrices. For fully-connected layers, the derivative after the weight matrix can be computed as $\mathcal{D}\mathbf{W} = \mathcal{D}\mathbf{s}(\bar{\mathbf{a}})^T$. With this, the block of the FIM corresponding to layer l is given by

$$\mathbf{F}_l = \mathbb{E}[\mathcal{D}\mathbf{s}(\mathcal{D}\mathbf{s})^T \otimes \bar{\mathbf{a}}(\bar{\mathbf{a}})^T]. \tag{2.2}$$

As convolutional layers share the weight tensor among the spatial positions, the derivative is a sum of outer products: $\mathcal{D}\mathbf{W}_{i,k} = \sum_{\mathbf{t}\in\mathcal{T}} \mathcal{D}\mathbf{s}_{\mathbf{t},i}\bar{\mathbf{a}}_{k,\mathbf{t}}$. Therefore, the FIM block for layer l can be formulated as

$$\mathbf{F}_l = \mathbb{E}[\sum_{\mathbf{t}\in\mathcal{T}} \sum_{\mathbf{t}'\in\mathcal{T}} \mathcal{D}\mathbf{s}_{\mathbf{t}}(\mathcal{D}\mathbf{s})_{\mathbf{t}'}^T \otimes \bar{\mathbf{a}}_{:,\mathbf{t}}(\bar{\mathbf{a}}_{:,\mathbf{t}'})^T]. \tag{2.3}$$

Kronecker-Factored Approximate Curvature. The *Kronecker-Factored Approximate Curvature* (K-FAC) approximates this FIM of a fully-connected layer [71] and a convolutional layer [38] by

$$\mathbf{F}_l \approx \mathbb{E}[\mathcal{D}\mathbf{s}(\mathcal{D}\mathbf{s})^T] \otimes \mathbb{E}[\bar{\mathbf{a}}(\bar{\mathbf{a}})^T] \quad \text{and} \quad \mathbf{F}_l \approx \mathbb{E}[(\mathcal{D}\mathbf{s})^T\mathcal{D}\mathbf{s}] \otimes \frac{1}{|\mathcal{T}|}\mathbb{E}[\bar{\mathbf{a}}(\bar{\mathbf{a}})^T],$$

respectively. In particular, K-FAC approximates the expected Kronecker product as a Kronecker product of expectations, which is not true in general but leads to Kronecker-factored blocks of the FIM. The Kronecker factorization enables the storage of two smaller matrices rather than a prohibitively large matrix [71]. However, these factorizations assume that the activations and the corresponding pre-activations are statistically independent, which is usually not met in practice [106]. Furthermore, additional assumptions like the independence of the first and second-order statistics of the spatial positions are used for convolutional layers, which might impair the approximation quality of the method.

Kronecker-Factored Optimal Curvature. Another tractable approximation of the FIM is the *Kronecker-factored Optimal Curvature* (K-FOC) which finds optimal Kronecker factors for each batch of data points and approximates both, the linear and the convolutional layer, as Kronecker product with two factors,

$$\mathbf{F}_l \approx \mathbf{L}_l \otimes \mathbf{R}_l.$$

It transforms the problem of finding optimal Kronecker-factors into a best rank-1-problem and solves it with a scalable version of the power method. The approximation is usually closer to the FIM than the approximation by K-FAC in terms of the Frobenius error. For more details on the K-FOC, see Appendix A.1.

Laplace Approximation for K-FAC and K-FOC. Given a block-diagonal Kronecker-factored precision matrix

$$H = \begin{pmatrix} \mathbf{L}_1 \otimes \mathbf{R}_1 & 0 & \dots & 0 \\ 0 & \mathbf{L}_2 \otimes \mathbf{R}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{L}_L \otimes \mathbf{R}_L \end{pmatrix},$$

the normal distribution of Equation 2.1 reduces to L independent matrix normal distributions

$$\mathcal{N}(\hat{\boldsymbol{\theta}}, \mathbf{F}^{-1}) = \prod_{l=1}^{L} \mathcal{MN}(\hat{\mathbf{W}}^l, \mathbf{L}_l, \mathbf{R}_l).$$

In the following, we assume a block-diagonal approximation of the FIM where each block $\mathbf{F}_l \in \mathbb{R}^{n_l \times n_l}$ corresponds to a layer $l \in [L]$. For fully-connected and convolutional layers, the dimensionality n_l is the size of the vectorized weight matrix, *i.e.* for fully-connected layers $n_l = d_l(d_{l-1} + 1)$ and for convolutional layers $n_l = c_l(c_{l-1}|\Delta^l| + 1)$.

2.4. PAC-Bayesian Bounds

Even though Bayesian neural networks were introduced in the Bayesian framework, *Probably Approximately Correct (PAC)-Bayesian bounds* introduce a frequentist method to bound the generalization of statistical functions like Bayesian neural networks [33]. PAC bounds aim to upper bound the risk, *i.e.* the expected loss on the true data distribution, with high probability using properties of the architecture and optimization of neural networks [115]. PAC-Bayesian bounds obtain similar bounds for Bayesian neural networks by comparing the posterior distribution with a dataset independent prior distribution [39].

Let $\mathcal{G} = \{g : \mathcal{X} \to \mathcal{Y} \mid g \text{ is measurable}\}$ be the set of hypotheses and a corresponding σ-algebra $\mathfrak{G}$ such that $(\mathcal{G}, \mathfrak{G})$ is a measurable space. Denote the set of probability distributions on this measurable space as

$$\mathcal{M} = \{\mu : \mathcal{G} \to [0, 1] \mid \mu \text{ is a probability distribution on } (\mathcal{G}, \mathfrak{G})\}.$$

Moreover, let $l : \mathcal{G} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. Then define the risk for $g \in \mathcal{G}$ as

$$\mathcal{L}_P^l(g) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P}\left[l(g, \mathbf{x}, \mathbf{y})\right]$$

and its empirical counterpart on the training data as

$$\hat{\mathcal{L}}_{\mathcal{D}}^l(g) = \frac{1}{N}\sum_{i=1}^{N} l(g, \mathbf{x}_i, \mathbf{y}_i).$$

Note that the neural network $f_{\boldsymbol{\theta}}$ as defined above is not in $\mathcal{G}$ because its output does not have to be in $\mathcal{Y}$. It only specifies the parameters of a distribution $p(\cdot|\mathbf{x}, f_{\boldsymbol{\theta}})$ over $\mathcal{Y}$. Nonetheless, for example, the function defined by $x \mapsto \arg\max_{y \in \mathcal{Y}} p(y|\mathbf{x}, f_{\boldsymbol{\theta}})$ is in $\mathcal{G}$.

Given a dataset independent prior distribution over the hypothesis set $\pi \in \mathcal{M}$, the PAC-Bayesian theory bounds the probability that the expected risk is large for hypotheses sampled from another probability distribution $\rho \in \mathcal{M}$ which is absolutely continuous w.r.t. π:

$$P_{D \sim P^N}\left(\forall \rho \in \mathcal{M},\ \rho \ll \pi :\ \mathbb{E}_{g \sim \rho}[\mathcal{L}_P^l(g)] \leq \delta(\rho, \pi, \mathcal{D}, \varepsilon)\right) \geq 1 - \varepsilon, \tag{2.4}$$

for $\varepsilon > 0$ and the PAC-Bayesian bound $\delta(\rho, \pi, \mathcal{D}, \varepsilon)$ [39].

The first bound was introduced by McAllester [72–75] for bounded loss functions.

Definition 2.4.1 (McAllester Bound [39])**.** Let $\varepsilon > 0$, $\rho, \pi \in \mathcal{M}$, $\rho \ll \pi$ and $l : \mathcal{G} \times \mathcal{X} \times \mathcal{Y} \to [0, 1]$, then

$$\delta(\rho, \pi, \mathcal{D}, \varepsilon) = \mathbb{E}_{g \sim \rho}[\hat{\mathcal{L}}_{\mathcal{D}}^l(g)] + \sqrt{\frac{\mathbb{KL}(\rho \| \pi) + \ln \frac{2\sqrt{N}}{\varepsilon}}{2N}} \tag{2.5}$$

is an upper bound for Equation 2.4.

The bound was improved for the error loss function by Catoni [13] when $\frac{\mathbb{KL}(\rho\|\pi)}{N}$ is large. The error loss function, or 0-1-loss, is defined as

$$\mathrm{er} : \mathcal{G} \times \mathcal{X} \times \mathcal{Y} \to \{0, 1\},\ \mathrm{er}(g, \mathbf{x}, \mathbf{y}) = \mathbb{1}[f(\mathbf{x}) \neq \mathbf{y}], \tag{2.6}$$

where $\mathbb{1}[s]$ is one if statement s is true and else zero.

Definition 2.4.2 (Catoni Bound [13])**.** Let $\varepsilon > 0$, $\rho, \pi \in \mathcal{M}$ and $\rho \ll \pi$, then

$$\delta(\rho, \pi, \mathcal{D}, \varepsilon) = \inf_{c > 0} \frac{1 - \exp(-c\mathbb{E}_{g \sim \rho}[\hat{\mathcal{L}}_{\mathcal{D}}^{\mathrm{er}}(g)] - \frac{\mathbb{KL}(\rho\|\pi) - \ln \varepsilon}{N})}{1 - \exp(-c)} \tag{2.7}$$

is an upper bound for Equation 2.4.

Remark. Note that in the PAC-Bayesian literature ρ is called posterior even though it is an arbitrary distribution that is dependent on the data [39]. To distinguish this from the posterior computed by Bayes' rule, we will explicitly write PAC-Bayes posterior if we do not use Bayes' rule.

The PAC-Bayes bounds depend mainly on two terms: the KL divergence of the PAC-Bayes prior and posterior and the empirical risk. Therefore, these bounds are small when the posterior is close to the prior and the loss on the training data is low. Thus, a good model should explain the training data well while depending not too much on it.

In PAC-Bayesian bounds, the prior and the posterior are both distributions over the functions and not over the weights like in Bayesian neural networks. Hence, to apply the PAC-Bayesian bounds for Bayesian neural networks, one needs to identify each weight sample with a sample in the function space. However, neural networks are not identifiable [11, 87]. Thus, multiple different weight vectors can explain the same function given by a neural network architecture. The KL-divergence is therefore smaller in function space than in weight space and one obtains an upper bound by considering the distributions over the weights [25].

In this thesis, we use the two bounds introduced above. Nonetheless, we find an approximate upper bound of the expected empirical error for Laplace approximation and can compute the KL-divergence in closed form for our models. Hence, all results of this thesis can directly be applied to other PAC-Bayesian bounds given that they depend on the expected empirical error and the KL-divergence of the posterior and the prior.

2.5. Priors

Now that we have introduced Bayesian neural networks and PAC-Bayesian bounds, this section summarizes how the prior is chosen for each of them in practice. Based on Bayes' rule, one can already see that the prior distribution has a great influence on the posterior, especially when only few data is available [30]. In the PAC-Bayesian theory, a well-specified prior can lead to better generalization bounds. On top of that, recent approaches propose that improvements in the prior are crucial for a better posterior [30, 112, 113]. Altogether, specifying useful prior distributions is of high priority. Still, most existing work uses uninformative prior distributions that use no information about the task and its properties [30]. In the following, common priors are introduced. In the first subsection, uninformative priors are introduced, followed by functional priors that specify priors in the function space and empirical priors that use data to learn a prior distribution.

2.5.1. Uninformative Priors

The most common prior for the weights of Bayesian neural network is an isotropic Gaussian around 0, *i.e.* $\pi = \mathcal{N}(0, \sigma^2 I)$ with $\sigma^2 > 0$ being the variance in each coordinate direction. In MAP estimation this prior corresponds to $\mathcal{L}_2$-regularization with weight decay $\gamma = \frac{1}{\sigma^2}$. Especially for variational inference, also other distributions from the exponential family are used. For instance, a matrix variate normal prior for each weight matrix is a common trade-off between correlations between weights and size of the covariance matrix [101] that is also used in this thesis. It is defined as follows:

Definition 2.5.1 (Matrix Variate Normal Distribution [41])**.** Let $\mathbf{L} \in \mathbb{R}^{m \times m}$, $\mathbf{R} \in \mathbb{R}^{n \times n}$ be positive semi-definite and $\mathbf{W}, \mathbf{M} \in \mathbb{R}^{m \times n}$. Then the density function of the *matrix*

variate normal distribution is given by

$$\begin{aligned}\mathcal{MN}(\mathbf{W}|\mathbf{M},\mathbf{L},\mathbf{R}) &= \frac{\exp(-\frac{1}{2}\operatorname{tr}(\mathbf{R}^{-1}(\mathbf{W}-\mathbf{M})^T\mathbf{L}^{-1}(\mathbf{W}-\mathbf{M})))}{(2\pi)^{\frac{mn}{2}}\det\mathbf{L}^{\frac{n}{2}}\det\mathbf{R}^{\frac{m}{2}}} \\ &= \mathcal{N}(\operatorname{vec}(\mathbf{W})|\operatorname{vec}(\mathbf{M}),\mathbf{L}\otimes\mathbf{R}),\end{aligned} \tag{2.8}$$

where $\otimes$ denotes the Kronecker product. Hence, it corresponds to a normal distribution with a Kronecker-factored covariance matrix and given $\mathbf{L} =: \mathbf{A}\mathbf{A}^T$ and $\mathbf{R} =: \mathbf{B}\mathbf{B}^T$, it can be sampled with

$$\begin{gathered}\mathbf{W}_{i,j} \sim \mathcal{N}(0,1) \quad \text{for } (i,j) \in [m]\times[n] \\ \mathbf{M} + \mathbf{A}\mathbf{W}\mathbf{B}^T \sim \mathcal{MN}(\mathbf{M},\mathbf{L},\mathbf{R}).\end{gathered}$$

Remark. Note that row vectorization is used to be consistent with PyTorch [84]. For column vectorization, the left and right factor need to be switched in the Kronecker product. Here, the matrices L and R correspond to the covariance matrices of the columns and rows, respectively. Altogether, matrix variate normal distributions allow correlations between variables without having to determine the whole covariance matrix [101].

A widespread method to increase the complexity of the prior distribution and to learn the parameters are hierarchical models. Here, the parameters of the prior are additionally modeled to follow a hyperprior distribution. Given a parameterized prior $\pi(\boldsymbol{\theta}|\psi)$, the hyperprior is a distribution over ψ. Common combinations are multivariate normal distributions as a prior with an inverse Wishart distribution over the covariance matrix [7] or horseshoe priors that consist of independent Gaussians for each parameter with half-Cauchy priors over the variance [34].

A common method to improve the uncertainty estimates in neural networks is applying dropout during inference. Dropout was originally introduced as a regularization method in which a subset of the weights is randomly set to zero during training to reduce dependence on each individual weight [45]. Gal & Ghahramani [31] were the first ones to use dropout during inference to generate multiple model samples to improve the uncertainty estimates of the model, called Monte Carlo (MC) dropout. From a Bayesian perspective, dropout can be seen as a hierarchical Bayesian neural network with a spike-and-slab prior on the weights which corresponds to a Gaussian prior with a Bernoulli hyperprior on the variance [78]. In this thesis, we compare our method against MC dropout.

2.5.2. Functional Priors

In general, it is difficult to specify special properties of the resulting Bayesian neural network like periodicity and invariance with a weight prior because of the complex interactions between the weights. Therefore, it was proposed to use prior distributions on the function defined by the Bayesian neural network. Similar to the priors in PAC-Bayesian theory, these priors are distributions over functions. Such priors often aim to encode

desired properties of the resulting function like the periodicity or the invariance to translations of the input. Most commonly a stochastic process prior is chosen that encodes the desired properties [30]. In this way, the resulting Bayesian neural network is regularized to be close to the prior, whereby the properties of the prior should be transferred to the Bayesian neural network. As the functional space is infinite-dimensional, approximations are needed to optimize the posterior weights. For this, either the functional prior can be mapped to a weight prior [29] or the objective can be approximated with a tractable heuristic [102]. Nonetheless, these priors are only considered for variational inference so far and suffer from stability issues [29, 102].

2.5.3. Empirical Priors

Another possibility to circumvent the manual specification of weight space priors is by using data to learn the prior, called empirical Bayes. In the beginning, empirical Bayes methods used hierarchical distributions and learned the parameters of the hyperprior with MLE. This is especially useful when the number of parameters for the hyperprior is much smaller than the number of parameters in the model and when the conjugate priors are chosen to directly optimize for the optimal hyperprior parameters [116]. More recent work use pre-trained models either on the same [57] or on related datasets [56] to specify the mean of a Gaussian prior distribution. The other parameters are then tuned by hand.

In the PAC-Bayesian setting, the PAC-Bayes prior should be close to the PAC-Bayes posterior without directly depending on the dataset. Nonetheless, it can depend on the data distribution. Hence, one approach is to split the training data into two sets, where one is used to learn the PAC-Bayes prior distribution and the other one for the PAC-Bayesian posterior [24, 83]. Another method is to learn the PAC-Bayes prior on the full training data but ensure that the dependence on each data sample is not too large, *e.g.* with differential privacy [26] or stability conditions [94].

2.6. Continual Learning

In the previous sections, the background on Bayesian neural networks and in particular on Laplace approximation is given. PBNN extend Bayesian neural networks to the continual learning setting. The remaining chapter reviews the basic concepts of continual learning that are needed for PBNN. *Continual learning* means that multiple tasks $\mathfrak{T}_1, \dots, \mathfrak{T}_T$ should be solved sequentially while only one task can be accessed at a time. Here, each task $\mathfrak{T}_t = (P_t, \mathcal{D}_t)$ consists of an unknown probability distribution over the domain $\mathcal{X}_t \times \mathcal{Y}_t$ and independent and identically distributed training samples $\mathcal{D}_t := \left((\mathbf{x}_i^{(t)}, \mathbf{y}_i^{(t)})\right)_{i=1}^{N_t} \sim P_t^{N_t}$. The goal is to transfer knowledge between tasks, *i.e.* the performance on each task should be better than using an individual network for that task. The main problem of neural networks in the context of continual learning is catastrophic forgetting which means that the performance on previously learned tasks is reduced by learning on new tasks [15].

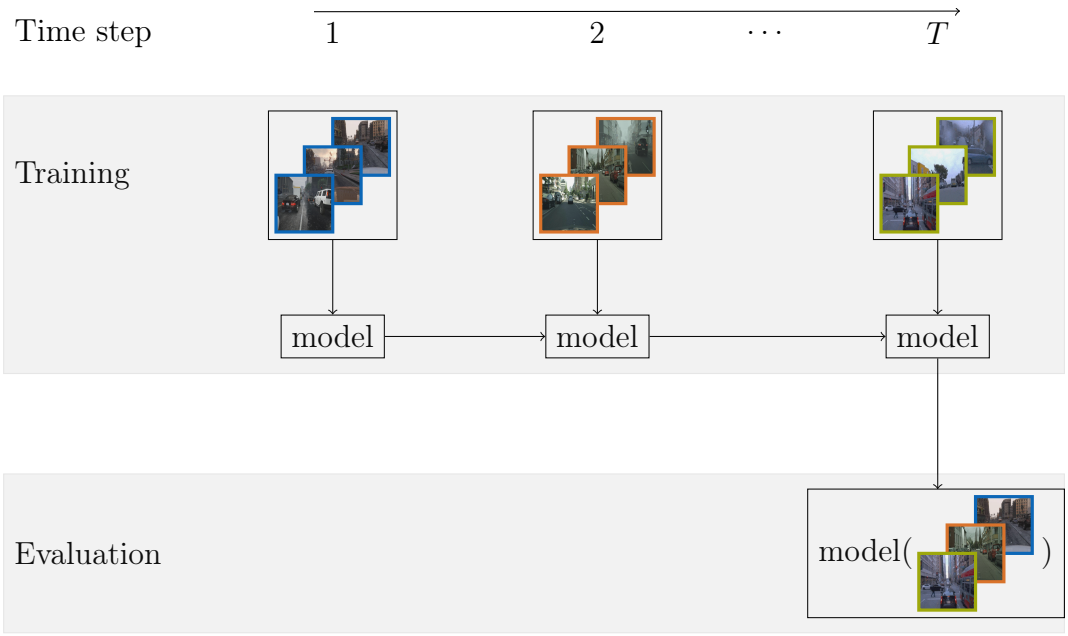


Figure 2.3.: In continual learning, multiple tasks should be solved sequentially while previous tasks can not be accessed. The resulting model is then evaluated jointly on all tasks. Image sources: [18, 90, 100]

Types of Continual Learning. In general, there are four types of continual learning depending on what is given during training and testing. All types usually assume the same input domain for all tasks, that is $\mathcal{X} := \mathcal{X}_t = \mathcal{X}_{t'}$ [47, 108]. The most common type is *task-incremental learning* [47], where the task id $t \in [T]$ is known during training and testing. This often comes along with different targets for different tasks, *i.e.* $\mathcal{Y}_t \neq \mathcal{Y}_{t'}$ for $t \neq t'$. In contrast, in *class-incremental continual learning* [47], task t is only known during the training and not during the testing phase. Moreover, all tasks usually share one target space, $\mathcal{Y} := \mathcal{Y}_t = \mathcal{Y}_{t'}$, but the distribution over the input and the target space is in general different, $P_t(\mathbf{y}) \neq P_{t'}(\mathbf{y})$ and $P_t(\mathbf{x}) \neq P_{t'}(\mathbf{x}')$ for $t \neq t'$, $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$. *Domain-incremental* is similar to class-incremental learning but one does not have to infer the task during testing as all tasks share the same target distribution, $P_t(\mathbf{y}) \neq P_{t'}(\mathbf{y})$. The arguably most complex type of continual learning is *task-agnostic* [47], where the task is not known during testing nor training. This thesis mainly concentrates on task-incremental continual learning as we are interested in including other related tasks in the training that have different targets.

Approaches. The approaches to solving continual learning can be classified into three categories – memory-based, regularization-based, and architecture-based methods [19, 91].

Memory-based methods keep a subset of samples from previous tasks or a model that can generate them [98]. The samples are replayed during training or used to constraint the model to fit older tasks. Prominent examples are iCaRL and GEM [19]. iCaRL [89] uses a subset of samples for each class to do class-incremental learning. In contrast, GEM [65] projects the gradient in each update step to a common space of the previous tasks to do

task-incremental learning. In general, memory-based methods achieve high performance on continual learning tasks. However, with an increasing number of tasks, either a huge number of samples needs to be kept in memory or each task is not represented enough by the given samples. Moreover, keeping raw samples in memory is often not possible because of privacy concerns especially in the medical domain [19].

Another approach to preventing catastrophic forgetting is to not change the important parameters of previous tasks too much in the current task. This is achieved by adding a regularization term that ensures that these parameters do not change too much, why these methods are called *regularization-based* methods. Prominent examples include Elastic Weight Consolidation (EWC) [55] and the Online Structured Laplace Approximation (OSLA) [93]. Both methods use the Laplace approximation to estimate the posterior of given a task which is then used as a prior in the next task. While EWC uses a diagonal approximation of the FIM, OSLA uses a Kronecker-factorization. Compared to memory-based approaches, regularization-based methods often require less memory and do not have the privacy issue of keeping raw samples in memory. Nevertheless, regularization-based methods usually do not achieve the performance of memory-based methods, especially for a lot of tasks [19].

The third method is *architecture-based*, which means that new model parameters are added or removed for each new task. A common example for this are *progressive neural networks* (PNN) [96]. For this, each task gets an individual copy of a base network that is trained on this task. When observing a new task, the old column is frozen. Moreover, parameterized lateral connections between previous columns and the current column are attached such that features from previous columns can be used. PNN and their variants are explained in more detail in Section 2.7. Compared to the other two approaches, architecture-based methods completely prevent catastrophic forgetting. However, they are mainly limited to task-incremental learning and have the disadvantage that the computational effort and memory grows directly with the number of tasks.

Even though this thesis uses a similar regularization and Laplace approximation as OSLA, we use it to generate scalable uncertainty estimates. Therefore, we need a flexible prior that is broad enough to find a good optimum with respect to the current tasks, but at the same time rigid enough to lead to a posterior which produces usable samples. Since with an increasing number of tasks, there is more information incorporated in the prior for regularization-based methods, it is a challenge to combine this information with a prior that can be used for sampling. Moreover, our main goal is to improve positive transfer from previous tasks. Therefore, we approach the continuous learning aspect with a progressive architecture similar to PNN which is able to completely mitigate catastrophic forgetting and enforces a transfer of knowledge from earlier tasks. Thus, PBNN can also be categorized as an architecture-based method.

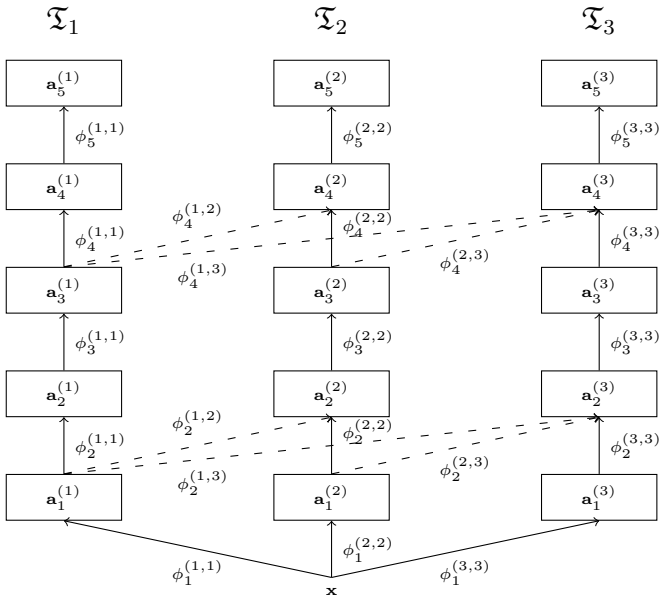


Figure 2.4.: Progressive neural networks (PNN) use weighted lateral connections to preceding columns to use previously learned features. Each column t is learned on the corresponding task $\mathfrak{T}_t$. All previous columns are frozen. The figure shows a PNN with five layers and the lateral connections $\mathfrak{L} = \{2, 4\}$.

2.7. Progressive Neural Networks

This work relies heavily on PNN to achieve continuous learning and, in particular, positive transfer from previous tasks. Therefore, PNN are discussed in more detail in this section. PNN [96] iteratively increase the network size for each new task that comes in. This is done by appending a copy of a base network – the new column – to the existing architecture. Additionally, lateral connections in the form of weighted layers are attached to the new column that uses the features from the previous column. Given a basic feedforward neural network architecture like in Section 2.1 and the previous columns corresponding to task $\mathfrak{T}_1, \ldots, \mathfrak{T}_{t-1}$, a PNN layer $l \in \mathfrak{L}$ that is in the set of lateral connections $\mathfrak{L} \subset [L]$ for column t is defined as

$$\mathbf{s}_l^{(t)} = \sum_{t'=1}^{t} \phi_l^{(t',t)}(\mathbf{a}_{l-1}^{(t')}), \quad \mathbf{a}_l^{(t)} = \sigma_l(\mathbf{s}_l^{(t)}), \tag{2.9}$$

where the superscript (t', t) denotes the connection from column t' to column t. For $l \notin \mathfrak{L}$ the layer is defined as usual by

$$\mathbf{s}_l^{(t)} = \phi_l^{(t,t)}(\mathbf{a}_{l-1}^{(t)}).$$

The dimensionality of the activations and pre-activations for layer $l \in [L-1]$ is the same for all columns and the activation function for layer l is also shared among tasks. Only the last layer can have a different number of outputs. Moreover, the same input $\mathbf{x} = \mathbf{a}_0^t = \mathbf{a}_0^{t'}$ is used for all columns.

Rusu *et al.* [96] established the use of so-called adapters that reduce the feature dimensionality and support the adaption of features from one column to the other. This is achieved by applying an additional hidden layer on the activation from the lateral connections. It is defined as

$$\mathbf{s}_l^{(t)} = \phi_l^{(t,t)}(\mathbf{a}_{l-1}^{(t)}) + \phi_{l,a}^{(t,t)}(\sigma_{l,a}(\sum_{t'=1}^{t-1} \phi_l^{(t',t)}(\alpha_l^{(t',t)}\mathbf{a}_{l-1}^{(t')}))),$$

where $\phi_{l,a}^{(t,t)}$ is an additional weighted layer, $\sigma_{l,a}$ is the activation function, and $\alpha_l^{(t',t)} \in \mathbb{R}$ is a small learnable scalar to adapt the scale of the individual features for the adapter. For convolutional layers, the dimensionality reduction is done with 1×1 convolutions for $\phi_l^{(t',t)}$.

Wang *et al.* [111] modified this method especially for transfer learning in vision problems. For this, they use the activations of the current layer instead of the ones from the previous layer, *i.e.*

$$\mathbf{s}_l^{(t)} = \phi_l^{(t,t)}(\mathbf{a}_{l-1}^{(t)}) + \sigma_{l,a}(\sum_{t'=1}^{t-1} \phi_l^{(t',t)}(\mathbf{a}_l^{(t')})).$$

In this thesis, the plain PNN definition as in Equation 2.9 is used. A five-layer PNN with three columns is shown in Figure 2.4. Here, the set of lateral connections is $\mathfrak{L} = \{2, 4\}$.

All variations start with a basic neural network for the first task. When a new column is added corresponding to task $\mathfrak{T}_t$, all previous columns and inter-column connections are frozen and only the column t together with incoming lateral connections are optimized. The optimization uses the output of column t, namely $\mathbf{a}_L^{(t)}$. As previous columns are not changed once their optimization is finished, the model does not forget the previous tasks. Moreover, previous features can be utilized to achieve a positive transfer.

3. Method

In this chapter, we introduce *Progressive Bayesian neural networks* (PBNN). This is a method that combines learned priors with progressive neural networks to employ previously learned features and distributions to improve the generalization and uncertainty estimates of the networks. In the first Section 3.1, the learning of a prior distribution and the automatic scaling of the curvatures is discussed in the context of transfer learning. The method is then extended to the continual learning setting with multiple sequentially incoming target tasks in Section 3.2.

3.1. Empirical Prior Learning

In this section, we propose *Empirical prior learning*, which extends deterministic pre-training to Bayesian neural networks. Especially in the field of computer vision, a lot of low-level feature extractors can be shared among different tasks [105, 119]. This is utilized in deterministic pre-training, where a network that is trained on a different dataset is used for the relevant task either for initialization or to generate feature embeddings where only the last few layers are fine-tuned [105]. Empirical prior learning follows this idea and specifies a prior around the pre-trained weights. Hence, the Bayesian neural network is pushed to be close to the pre-trained network distribution instead of fixing it to the given weights like in deterministic pre-training. Moreover, similar to randomly initialized priors, the learned prior breaks symmetries that come from the non-identifiability of weights in neural networks. This is expected to improve the PAC-Bayesian bounds [25]. Finally, it provides an alternative to empirical PAC-Bayesian methods that use one part of the training data to learn the prior and the other part for the posterior [24, 83]. Altogether, our empirically learned prior introduces a Bayesian version of deterministic pre-training that allows more flexibility than fine-tuning, is expected to break symmetries, and can use the whole training data to learn the posterior.

Setting. In this section, we consider a transfer learning setting. Thus, we have the source task $\mathfrak{T}_0$ and the target task $\mathfrak{T}_1$, each consisting of a set of samples $\mathcal{D}_t$ from a probability distribution P_t, $t \in \{0, 1\}$, defined similarly as in Section 2.6. The goal is to use task $\mathfrak{T}_0$ to improve the accuracy and uncertainty estimates on task $\mathfrak{T}_1$. The performance on $\mathfrak{T}_0$ is not considered because the first task is only used to learn the prior.

Main Idea. The main idea behind empirical prior learning is to learn a prior distribution from another dataset $\mathcal{D}_0$. We realize this with Laplace approximation. Hence, we approximate the posterior given $\mathcal{D}_0$ and use this distribution as the prior for the relevant task. This concept is explained in Section 3.1.1 and Section 3.1.2. To make Laplace approximation practical, the FIM or the precision matrix of the prior are commonly scaled. We aim to improve this scaling. For this, we introduce individual scales for the FIM and prior precision matrix of each layer and scale them using PAC-Bayesian bounds. Because the PAC-Bayesian bounds and in particular the expected empirical error can not be optimized directly, we derive an approximate upper bound that can easily be optimized. For the approximate upper bound, we use that a scaled version of the negative log-likelihood is an upper bound of the error function. This is combined with the Taylor approximation around the MAP weights with the FIM instead of the Hessian to obtain the approximate upper bound. The automatic curvature scaling is derived in detail in Section 3.1.3. Overall, the empirically learned prior with curvature scaling aims to improve the posterior with a better specified prior and smaller PAC-Bayesian bounds.

3.1.1. Prior

The empirically learned prior π corresponds to the posterior over the parameters given the source dataset $\mathcal{D}_0$,

$$\pi(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta}|\mathcal{D}_0) \propto p(\mathcal{D}_0|f_{\boldsymbol{\theta}})p(\boldsymbol{\theta}).$$

As it is usually infeasible to compute this posterior exactly, we use its Laplace approximation. This boils down to computing the MAP estimate with $\mathcal{L}_2$ weight decay $\gamma > 0$,

$$\begin{aligned}\hat{\boldsymbol{\theta}}^{(0)} &\in \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}_0) \\ &= \arg\min_{\boldsymbol{\theta}} -\ln p(\mathcal{D}_0|f_{\boldsymbol{\theta}}) + \frac{\gamma}{2}\|\boldsymbol{\theta}\|_2^2.\end{aligned}$$

This also makes it possible to use any pre-trained non-Bayesian model which was obtained with MAP estimation. In the next step, an approximation $\mathbf{F}^{(0)}$ of the FIM needs to be computed. Altogether, the empirically learned prior is then defined as the Laplace-approximated posterior around the MAP parameters

$$\pi = \mathcal{N}(\hat{\boldsymbol{\theta}}^{(0)}, \tau(\mathbf{F}^{(0)} + \gamma\mathbf{I})^{-1}), \tag{3.1}$$

where $\tau > 0$ is the temperature scaling. Here, the $\mathcal{L}_2$ weight decay γ determines the strength of the regularization and by adding a multiple of the identity matrix to the FIM, the precision matrix becomes positive definite. The temperature scaling τ is a commonly used heuristic to improve the approximation quality of Bayesian neural networks [112]. For Laplace approximation, choosing a small temperature scaling produces samples that are closer to the mean [64].

3.1.2. Posterior

Next, we aim to incorporate the learned prior around the pre-trained weights to compute the posterior on the relevant dataset $\mathcal{D}_1$. In contrast to an isotropic Gaussian prior, this prior pushes the weights close to already usable feature extractors. Moreover, it provides more flexibility than fixing the pre-trained weights like in fine-tuning as the posterior can deviate from the prior. Using the learned prior π, the posterior given dataset $\mathcal{D}_1$ can be computed as

$$p(\boldsymbol{\theta}|\mathcal{D}_1) \propto p(\mathcal{D}_1|f_{\boldsymbol{\theta}})\pi(\boldsymbol{\theta}).$$

Again, Laplace approximation is used to obtain a feasible distribution ρ. Hence, a MAP estimate,

$$\begin{aligned}\hat{\boldsymbol{\theta}}^{(1)} &\in \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}_1) \\ &= \arg\min_{\boldsymbol{\theta}} -\ln p(\mathcal{D}_1|f_{\boldsymbol{\theta}}) - \ln\left(\mathcal{N}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}^{(0)}, \tau(\mathbf{F}^{(0)} + \gamma\mathbf{I})^{-1})\right) \\ &= \arg\min_{\boldsymbol{\theta}} -\ln\left(p(\mathcal{D}_1|f_{\boldsymbol{\theta}})\right) + \frac{1}{2\tau}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}^{(0)})^T(\mathbf{F}^{(0)} + \gamma\mathbf{I})(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}^{(0)}),\end{aligned}$$

and an approximation of the FIM, $\mathbf{F}^1$, around this mode needs to be determined. The resulting tractable posterior is then

$$\rho = \mathcal{N}(\hat{\boldsymbol{\theta}}^1, \tau(\mathbf{F}^{(1)} + \mathbf{F}^{(0)} + \gamma\mathbf{I})^{-1}). \tag{3.2}$$

Instead of introducing a second temperature scaling for the posterior, we use the same temperature scaling as for the prior and weight the FIM and the precision matrix of the prior with the curvature scaling in Section 3.1.3.

For a diagonal or block-diagonal approximation of the FIM, the precision matrix

$$\tilde{\mathbf{F}}^{(1)} = \mathbf{F}^{(1)} + \mathbf{F}^{(0)} + \gamma\mathbf{I} \tag{3.3}$$

can be computed and inverted exactly. In contrast, the sum of multiple Kronecker-factored matrices is usually not Kronecker-factored. Therefore, we use the power method from Algorithm 2 to obtain a Kronecker-factored approximation of the precision matrix,

$$\tilde{\mathbf{F}}^{(1)} = \tilde{\mathbf{L}}^{(1)} \otimes \tilde{\mathbf{R}}^{(1)} \approx \mathbf{F}^{(1)} + \mathbf{F}^{(0)} + \gamma\mathbf{I}. \tag{3.4}$$

With this, one can exploit that the inverse of a Kronecker product is the same as the Kronecker product of the inverses [71].

Altogether, this method combines a prior that is learned from another dataset [93] with Bayesian inference for scalable Laplace approximation [92] and applies it to transfer learning. More generally, this method also corresponds to Bayesian online learning [81], where the belief over the parameters $\boldsymbol{\theta}$ is updated with each new dataset observed even though Bayesian online learning usually assumes the same data distribution for new incoming

data. Nonetheless, especially as we are mainly concerned with the performance on task $\mathfrak{T}_1$, the prior should not be too strict. Common strategies to change the broadness of distributions in Laplace approximation is scaling the precision matrices of the prior [55] or the FIM [92, 93]. In this thesis, we aim to find these scales for the precision matrix of the prior (Equation 3.1) and the posterior (Equation 3.2) automatically for each diagonal block of the precision matrix by optimizing PAC-Bayesian bounds. Hence, for task $\mathfrak{T}_0$, each block of the precision matrix of π corresponding to layer l is a linear combination of its FIM approximation and the prior curvature,

$$\tilde{\mathbf{F}}_l^{(0)} = \beta_l^{(0)} \mathbf{F}_l^{(0)} + \alpha_l^{(0)} \gamma \mathbf{I}.$$

Similarly, the precision matrix of the posterior ρ from Equation 3.3 and Equation 3.4 is given by

$$\tilde{\mathbf{F}}_l^{(1)} = \beta_l^{(1)} \mathbf{F}_l^{(1)} + \alpha_l^{(1)} \tilde{\mathbf{F}}_l^{(0)}.$$

This means that the precision matrix of each layer is scaled individually which should lead to lower PAC-Bayesian bounds and a better approximated posterior. While diagonal and block-diagonal approximations of this linear combination can be computed and inverted in closed form, we use the power method to obtain an estimate of the linear combination for the Kronecker-factored approximations. The upcoming Section 3.1.3 explains how $\alpha_l^{(t)}$ and $\beta_l^{(t)}$ for $l \in [L], t \in \{0, 1\}$ can be found using PAC-Bayesian bounds and Laplace approximation.

3.1.3. Curvature Scaling

As the curvature scaling is utilized both in the computation of the empirical prior and the posterior, we now consider the general case of having a prior $\pi = \mathcal{N}(\tilde{\boldsymbol{\theta}}, \tau \tilde{\mathbf{F}}^{-1})$, a dataset $\mathcal{D} = ((\mathbf{x}_i, \mathbf{y}_i))_{i=1}^N \in (\mathcal{X} \times \mathcal{Y})^N$ and a posterior distribution given the dataset $\rho = \mathcal{N}(\hat{\boldsymbol{\theta}}, \tau \hat{\mathbf{F}}^{-1})$, where each diagonal block l of the precision matrix is given by $\hat{\mathbf{F}}_l = \beta_l \mathbf{F}_l + \alpha_l \tilde{\mathbf{F}}_l$ and $\mathbf{F}$ is an approximation of the FIM on $\mathcal{D}$. The goal is now to find good values for β_l and α_l. When computing the empirical prior with dataset $\mathcal{D}_0$, $\tilde{\mathbf{F}} = \gamma \mathbf{I}$ and $\tilde{\boldsymbol{\theta}} = 0$ and for the posterior given dataset $\mathcal{D}_1$, one would choose $\tilde{\mathbf{F}} = \tilde{\mathbf{F}}^{(0)}$ and $\tilde{\boldsymbol{\theta}} = \hat{\boldsymbol{\theta}}^{(0)}$.

The curvature scales should handle the weighting of the prior for the current task. In particular, we aim to minimize the expected prediction error on the true data distribution with PAC-Bayesian bounds. As we study the 0-1-loss on the true data distribution, we only consider classification problems, *i.e.* $|\mathcal{Y}| < \infty$. For this, we use the hypothesis that chooses the most probable class, $g(x) = \arg\max_{\mathbf{y}' \in \mathcal{Y}} p(\mathbf{y}' | \mathbf{x}, f_{\boldsymbol{\theta}})$, given the neural network $f_{\boldsymbol{\theta}}$. Hence, with this correspondence, we use $f_{\boldsymbol{\theta}}$ and g interchangeably. To determine the PAC-Bayesian bounds from Section 2.4 using the error loss function, one needs to compute the expected empirical error on the training data,

$$\mathbb{E}_{\boldsymbol{\theta} \sim \rho} \left[\mathcal{L}_{\mathcal{D}}^{\mathrm{er}}(f_{\boldsymbol{\theta}}) \right] = \mathbb{E}_{\boldsymbol{\theta} \sim \rho} \left[\frac{1}{N} \sum_{i=1}^{N} \mathrm{er}(f_{\boldsymbol{\theta}}, \mathbf{x}_i, \mathbf{y}_i) \right],$$

and the KL-divergence between the prior and the posterior,

$$\mathbb{KL}(\rho\|\pi) = \mathbb{KL}(\mathcal{N}(\hat{\boldsymbol{\theta}}, \tau\hat{\mathbf{F}}^{-1})\|\mathcal{N}(\tilde{\boldsymbol{\theta}}, \tau\tilde{\mathbf{F}}^{-1})). \tag{3.5}$$

Moreover, both terms need to be optimized jointly with respect to the curvature scales.

Expected Empirical Error. The first term is only indirectly influenced by the curvature scaling as the samples drawn from the posterior depend on the precision matrix which itself depends on the scales. Moreover, the derivative of the error function is 0 almost everywhere, which makes it even harder to directly optimize this term. Therefore, we instead minimize an approximate upper bound of the expected empirical error on the training data utilizing on the one hand that a scaled data log-likelihood is an upper bound of the error function and on the other hand that we can approximate the data log-likelihood by its Taylor approximation similarly as in the Laplace approximation.

First, we derive the upper bound of the error function:

Lemma 3.1.1. *Let* $f : \mathcal{X} \to \Omega_L$, $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$ *and*

$$\mathrm{er}(f, \mathbf{x}, \mathbf{y}) = \mathbb{1}[\arg\max_{\mathbf{y}' \in \mathcal{Y}} p(\mathbf{y}'|\mathbf{x}, f) \neq \mathbf{y}],$$

then

$$\mathrm{er}(f, \mathbf{x}, \mathbf{y}) \leq -\frac{\ln p(\mathbf{y}|\mathbf{x}, f)}{\ln 2}.$$

Proof. The proof examines the case of a correct prediction first and of a wrong prediction second. For the correct prediction, the error is 0 and the bound reduces to the negative log-likelihood being larger than 0. In the case of a wrong prediction, we use that there is a class with a higher probability and that the correct and the most probable class have together a probability that can be bounded by 1 from above.

First, consider the case of a correct classification, which means that y has the largest probability: $\arg\max_{\mathbf{y}' \in \mathcal{Y}} p(\mathbf{y}'|\mathbf{x}, f) = \mathbf{y}$. Then, $\mathrm{er}(f, \mathbf{x}, \mathbf{y}) = 0$. As $p(\mathbf{y}|\mathbf{x}, f)$ is a discrete probability, $p(\mathbf{y}|\mathbf{x}, f) \leq 1$ and hence, by taking the negative logarithm on both sides, $-\frac{\ln p(\mathbf{y}|\mathbf{x}, f)}{\ln 2} \geq 0 = \mathrm{er}(f, \mathbf{x}, \mathbf{y})$.

Now, let $\arg\max_{\mathbf{y}' \in \mathcal{Y}} p(\mathbf{y}'|\mathbf{x}, f) \neq \mathbf{y}$. Therefore, $\mathrm{er}(f, \mathbf{x}, \mathbf{y}) = 1$ and there exists a $\mathbf{y}' \in \mathcal{Y}$ such that $p(\mathbf{y}'|\mathbf{x}, f) \geq p(\mathbf{y}|\mathbf{x}, f)$. Additionally, we have that $1 \geq p(\mathbf{y}'|\mathbf{x}, f) + p(\mathbf{y}|\mathbf{x}, f) \geq 2p(\mathbf{y}|\mathbf{x}, f)$. This can be written as $p(\mathbf{y}|\mathbf{x}, f) \leq \frac{1}{2}$. Due to the monotony of the logarithm, we can take the natural logarithm on both sides and divide by $-\ln 2$ to obtain that $-\frac{\ln p(\mathbf{y}|\mathbf{x}, f)}{\ln 2} \geq 1 = \mathrm{er}(f, \mathbf{x}, \mathbf{y})$.

Consequently, $-\frac{\ln p(\mathbf{y}|\mathbf{x}, f)}{\ln 2}$ is an upper bound of $\mathrm{er}(f, \mathbf{x}, \mathbf{y})$. □

Remark. We can see from the proof that the bound is tighter when the correct class has a high likelihood. Hence, the bound will be best for good performing models, while it might be loose when the negative data log-likelihood is large.

Lemma 3.1.1 gives an upper bound of the error function in terms of the negative log-likelihood. Therefore, the expected empirical error can be upper bounded by the scaled negative data log-likelihood. In the next step, we approximate the data log-likelihood by its second-order Taylor polynomial with a FIM approximation instead of the Hessian similar to Laplace approximation. Altogether, the final bound is

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\theta}\sim\rho}\left[\frac{1}{N}\sum_{i=1}^{N}\mathrm{er}(f_{\boldsymbol{\theta}},\mathbf{x}_i,\mathbf{y}_i)\right] &\leq \mathbb{E}_{\boldsymbol{\theta}\sim\rho}\left[\frac{1}{N}\sum_{i=1}^{N}-\frac{\ln p(\mathbf{y}_i|\mathbf{x}_i,f_{\boldsymbol{\theta}})}{\ln 2}\right] \\
&= \frac{1}{N\ln 2}\mathbb{E}_{\boldsymbol{\theta}\sim\rho}\left[-\ln p(\mathcal{D}|f_{\boldsymbol{\theta}})\right] \\
&\approx \frac{1}{N\ln 2}\mathbb{E}_{\boldsymbol{\theta}\sim\rho}\left[-\ln p(\mathcal{D}|f_{\hat{\boldsymbol{\theta}}})-\mathcal{D}\boldsymbol{\theta}^T(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})+\frac{1}{2}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})^T\mathbf{F}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})\right] \qquad (3.6)\\
&= \frac{1}{N\ln 2}\left(-\ln p(\mathcal{D}|f_{\hat{\boldsymbol{\theta}}})+\frac{1}{2}\mathbb{E}_{\boldsymbol{\theta}\sim\rho}\left[(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})^T\mathbf{F}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})\right]\right) \qquad (3.7)\\
&= \frac{1}{N\ln 2}\left(-\ln p(\mathcal{D}|f_{\hat{\boldsymbol{\theta}}})+\frac{\tau}{2}\sum_{l\in[L]}\mathrm{tr}\left(\mathbf{F}_l(\beta_l\mathbf{F}_l+\alpha_l\tilde{\mathbf{F}}_l)^{-1}\right)\right) \\
&= \frac{1}{N\ln 2}\left(-\ln p(\mathcal{D}|f_{\hat{\boldsymbol{\theta}}})+\frac{\tau}{2}\sum_{l\in[L]}\sum_{\lambda\in\Lambda(\mathbf{F}_l\tilde{\mathbf{F}}_l^{-1})}\frac{1}{\beta_l+\alpha_l\lambda^{-1}}\right), \qquad (3.8)
\end{aligned}$$

where Lemma 3.1.1 is used in the first equation and the Taylor approximation in the third. Equation 3.7 uses that $\ln p(\mathcal{D}|f_{\hat{\boldsymbol{\theta}}})$ is independent of $\boldsymbol{\theta}$ and $\mathbb{E}_{\boldsymbol{\theta}\sim\rho}\left[\mathcal{D}\boldsymbol{\theta}^T(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})\right] = \mathcal{D}\boldsymbol{\theta}^T(\mathbb{E}_{\boldsymbol{\theta}\sim\rho}[\boldsymbol{\theta}]-\hat{\boldsymbol{\theta}}) = \mathcal{D}\boldsymbol{\theta}^T(\hat{\boldsymbol{\theta}}-\hat{\boldsymbol{\theta}}) = 0$. Here, we define $\Lambda(\mathbf{F}_l\tilde{\mathbf{F}}_l^{-1})$ to be the set of eigenvalues of the matrix $\mathbf{F}_l\tilde{\mathbf{F}}_l^{-1}$. This means that we formulate the approximate upper bound of the expected empirical error by the data log-likelihood at the MAP estimate and the eigenvalues of the product of the FIM and the covariance matrix of the prior.

KL-Divergence. As the prior and the posterior are both multivariate Gaussians, the KL-divergence [23] of Equation 3.5 can be computed by

$$\begin{aligned}
&\mathbb{KL}(\mathcal{N}(\hat{\boldsymbol{\theta}},\tau\hat{\mathbf{F}}^{-1})\|\mathcal{N}(\tilde{\boldsymbol{\theta}},\tau\tilde{\mathbf{F}}^{-1})) = \frac{1}{2}\left(\mathrm{tr}(\tilde{\mathbf{F}}\hat{\mathbf{F}}^{-1})-n-\ln\frac{\det\tilde{\mathbf{F}}}{\det\hat{\mathbf{F}}}+\frac{1}{\tau}(\hat{\boldsymbol{\theta}}-\tilde{\boldsymbol{\theta}})^T\tilde{\mathbf{F}}(\hat{\boldsymbol{\theta}}-\tilde{\boldsymbol{\theta}})\right) \\
&= \frac{1}{2}\left(\sum_{l\in[L]}\mathrm{tr}(\tilde{\mathbf{F}}_l(\beta_l\mathbf{F}_l+\alpha_l\tilde{\mathbf{F}}_l)^{-1})-n_l-\ln\frac{\det\tilde{\mathbf{F}}_l}{\det(\beta_l\mathbf{F}_l+\alpha_l\tilde{\mathbf{F}}_l)}+\frac{1}{\tau}(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)^T\tilde{\mathbf{F}}_l(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)\right) \\
&= \frac{1}{2}\left(\sum_{l\in[L]}\left(\sum_{\lambda\in\Lambda(\mathbf{F}_l\tilde{\mathbf{F}}_l^{-1})}\frac{1}{\beta_l\lambda+\alpha_l}-1-\ln(\beta_l\lambda+\alpha_l)\right)+\frac{1}{\tau}(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)^T\tilde{\mathbf{F}}_l(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)\right). \qquad (3.9)
\end{aligned}$$

Therefore, both terms necessary for the PAC-Bayesian bounds can be computed with the eigenvalues of the product between the FIM of the likelihood and the inverse of the precision matrix of the prior, $\mathbf{F}_l\tilde{\mathbf{F}}_l^{-1}$. For common approximations of the FIM as a diagonal or a Kronecker-factored block-diagonal matrix, the eigenvalues can be obtained without computing the full FIM. When both $\mathbf{F}_l$ and $\tilde{\mathbf{F}}_l$ are diagonal, the desired eigenvalues are just the diagonal elements of the product, while for Kronecker-factored matrices all eigenvalues are given by the product of one eigenvalue from each factor, *i.e.* $\Lambda(\mathbf{L}\otimes\mathbf{R}) = \{\lambda^{\mathbf{L}}\lambda^{\mathbf{R}}|\lambda^{\mathbf{L}}\in\Lambda(\mathbf{L}),\lambda^{\mathbf{R}}\in\Lambda(\mathbf{R})\}$.

Additionally, the quadratic terms in Equation 3.9 involve only minor computational effort for the diagonal and Kronecker-factored approximation. With a diagonal $\tilde{\mathbf{F}}_l$, the quadratic term can be computed with two element-wise multiplications of vectors of size n_l with $(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)^T\tilde{\mathbf{F}}_l(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l) = (\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)\odot\mathrm{diag}(\tilde{\mathbf{F}}_l)\odot(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)$, where $\mathrm{diag}(\tilde{\mathbf{F}}_l)$ is the vector containing the diagonal elements of $\tilde{\mathbf{F}}_l$. The complexity for Kronecker-factored matrices boils down to two matrix-matrix products with the Kronecker-factors and one element-wise product:

$$\begin{aligned}(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)^T\tilde{\mathbf{F}}_l(\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l) &= (\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)^T(\mathbf{L}\otimes\mathbf{R})(\hat{\mathbf{W}}^l-\tilde{\mathbf{W}}^l)\\ &= (\hat{\boldsymbol{\theta}}_l-\tilde{\boldsymbol{\theta}}_l)\odot\mathrm{vec}(\mathbf{L}(\hat{\mathbf{W}}^l-\tilde{\mathbf{W}}^l)\mathbf{R}),\end{aligned}$$

where $\hat{\mathbf{W}}^l$ and $\tilde{\mathbf{W}}^l$ are the weight matrices corresponding to $\hat{\boldsymbol{\theta}}_l$ and $\tilde{\boldsymbol{\theta}}_l$, respectively.

Altogether, this section so far introduced a tractable approximate upper bound of the PAC-Bayesian bounds, that can directly be optimized with respect to the curvature scales after computing the FIM approximation.

Approaches. The method so far consists of finding a MAP estimate for the current posterior, then computing an approximation of the FIM and merging the FIM with the prior using curvature scaling. This method combines a Bayesian method, namely the MAP estimation to obtain the optimal parameters, with the frequentist PAC-Bayesian bounds to improve the generalization. For completeness, we compare this *combined approach* against a fully Bayesian approach and a frequentist approach.

In the *fully Bayesian approach*, all curvature scales are set equal to one. This corresponds to Bayesian online learning, where the curvatures are not scaled at all.

In contrast, the *frequentist method* also searches for the optimal weights in terms of the PAC-Bayesian bounds instead of maximizing the posterior probability with MAP estimation. This has the goal of further minimizing the PAC-Bayesian bounds also with the choice of the parameters and not only with the curvature scaling. Nonetheless, as the FIM is only available after the weights are found, we neglect the terms that depend on the curvature. For the McAllester Bound, this results in the optimization problem

$$\min_{\boldsymbol{\theta}} -\frac{\ln p(\mathcal{D}|f_{\boldsymbol{\theta}})}{\ln(2)N} + \sqrt{\frac{\frac{1}{2\tau}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^T\tilde{\mathbf{F}}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}) + \ln(\frac{2N}{\varepsilon})}{2N}}. \tag{3.10}$$

The Catoni Bound is difficult to optimize with variations of stochastic gradient descent because of the exponential in the objective function. Nonetheless, when the Catoni scale

Table 3.1.: The different approaches used in this thesis.

Approach	Weight optimization	Curvature optimization
Fully Bayesian	MAP	$\alpha_l = \beta_l = 1$
Combined	MAP	McAllester (Equation 2.5)
	MAP	Catoni (Equation 2.7)
Frequentist	McAllester (Equation 3.10)	McAllester (Equation 2.5)
	Catoni (Equation 3.11, 3.12)	Catoni (Equation 2.7)

$c > 0$ is fixed, minimizing the upper bound of the Catoni Bound is equivalent to calculating

$$\min_{\boldsymbol{\theta}} -c\frac{\ln p(\mathcal{D}|f_{\boldsymbol{\theta}})}{\ln(2)N} + \frac{\frac{1}{2\tau}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^T\tilde{\mathbf{F}}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}) - \ln\varepsilon}{N}. \tag{3.11}$$

Therefore, we heuristically alternate between optimizing $\boldsymbol{\theta}$ and c in practice, where the optimization after the Catoni scale is done using the objective

$$\min_{c>0} \frac{1-\exp(c\frac{\ln p(\mathcal{D}|f_{\boldsymbol{\theta}})}{\ln(2)N} - \frac{\frac{1}{2\tau}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^T\tilde{\mathbf{F}}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})-\ln\varepsilon}{N})}{1-\exp(-c)}. \tag{3.12}$$

While the Catoni bound is in general tighter, the optimization process is more difficult compared to the McAllester bound. After the optimization of the optimal parameters, the precision matrix is chosen with the curvature scaling as a linear combination of the prior precision matrix and the FIM. Different from the combined approach, this is not a valid Laplace approximation as the first-order term of the corresponding Taylor approximation does not vanish in general because the parameters are not a local optimum of the posterior. Nonetheless, this method aims to improve the generalization and in particular the generalization bounds better compared to the other two approaches.

In total, we therefore have three different approaches to compute the optimal parameters and the scales of the curvature – a fully Bayesian, a frequentist, and a combined approach. An overview of the different approaches is given in Table 3.1

3.2. Progressive Bayesian Neural Networks

In the following, the empirical prior learning is extended to continual learning. This is in particular relevant for domains where many related datasets are available and one aims to incorporate them into the training. In contrast to continual learning approaches building on top of Bayesian online learning [55, 93] we do not reduce forgetting by putting all previous information in the prior distribution but by an architecture-based approach. This has the advantage that catastrophic forgetting is completely mitigated and we can choose a flexible prior even after observing a lot of different tasks. Altogether, PBNN transfer the abilities of progressive neural networks to use previously learned feature embeddings in a continual learning fashion to Bayesian deep learning.

Setting. In the following, we combine the notation from empirical prior learning (Section 3.1) and continual learning (Section 2.6 and Section 2.7). Hence, the goal is to solve tasks $\mathfrak{T}_1, \dots, \mathfrak{T}_T$ sequentially, where $T \in \mathbb{N}$. Moreover, an additional task $\mathfrak{T}_0$ is given to compute a prior distribution for which the performance is not relevant. Each of the tasks consists of an unknown data distribution and a dataset sampled from this distribution, $\mathfrak{T}_t = (P_t, \mathcal{D}_t)$ for $t \in [T]_0$. All tasks share a common input domain $\mathcal{X}$ but can have different finite target spaces $\mathcal{Y}_t$. The dataset is given by $\mathcal{D}_t = \left((\mathbf{x}_i^{(t)}, \mathbf{y}_i^{(t)})\right)_{i=1}^{N_t} \sim P_t^{N_t}$ with $N_t \in \mathbb{N}$ samples. In total, the problem is a task-incremental continual learning problem with an additional prior task.

In practice, the prior task could be a common dataset that is used for deterministic pre-training like ImageNet [20] to generate broadly usable feature embeddings and corresponding weight distributions. Apart from the continual learning application, one can also think of incorporating simulation data and other relevant tasks by using PBNN. These could then be used as the early tasks $\mathfrak{T}_1, \mathfrak{T}_2, \dots$ to generate very specific feature embeddings that can be utilized by subsequent tasks. Figure 1.1 shows an example of such a problem setting.

Main Idea. PBNN incorporate multiple related tasks in the training of a Bayesian neural network that uses the empirically learned prior with curvature scaling. For this, we use PNN without adapter networks and transform them to Bayesian neural networks. Hence, a prior needs to be specified for the main columns and the lateral connections. For the main column, we select the empirically learned prior and for the lateral connections, the posterior from which the lateral connection originates is used. Therefore, both priors were learned on their respective incoming feature distribution. Altogether, PBNN allow using multiple tasks in combination with the empirically learned prior and curvature scaling to jointly improve the performance and the uncertainty estimation.

Notation. Before observing the first relevant task $\mathfrak{T}_1$, PBNN compute the empirically learned prior $\mathcal{N}(\hat{\boldsymbol{\theta}}^{(0)}, \tau(\tilde{\mathbf{F}}^{(0)})^{-1})$ given the dataset $\mathcal{D}_0$ as described in Section 3.1. In the next step, a progressive neural network is iteratively built for each task. In contrast to Rusu *et al.* [96], we additionally divide the pre-activations of the lateral connections by the number of incoming connections to improve the stability:

$$\mathbf{s}_l^{(t)} = \frac{1}{t} \sum_{t'=1}^{t} \phi_l^{(t',t)}(\mathbf{a}_{l-1}^{(t')}).$$

With this, the scale of the pre-activations does not increase for later columns. We denote the lateral connections by a superscript (t', t). Hence, the vectorized weight and the corresponding prior and posterior for a layer l, from column t' to t are written as $\boldsymbol{\theta}_l^{(t',t)}$, $\rho_l^{(t',t)}$, and $\pi_l^{(t',t)}$, respectively. When $t' < t$, the connection is between different columns while $t' = t$ means that the weight is in the main column. Moreover, we use the superscript (t) to denote all connections that are incoming to column t. This also includes the main

	Weight	**Lateral**	**Column**	**Full**
Weight	$\boldsymbol{\theta}_l^{(t',t)}$	$\boldsymbol{\theta}^{(t',t)} = \text{vec}(\boldsymbol{\theta}_l^{(t',t)})_{l\in[L]}$	$\boldsymbol{\theta}^{(t)} = \text{vec}(\boldsymbol{\theta}^{(t',t)})_{t'\in[t]}$	$\boldsymbol{\theta}^{(\leq t)} = \text{vec}(\boldsymbol{\theta}^{(t')})_{t'\in[t]}$
Prior	$\pi_l^{(t',t)}$	$\pi^{(t',t)} = \prod_{l=1}^{L} \pi_l^{(t',t)}$	$\pi^{(t)} = \prod_{t'=1}^{t} \pi^{(t',t)}$	$\pi^{(\leq t)} = \prod_{t'=1}^{t} \pi^{(t')}$
Posterior	$\rho_l^{(t',t)}$	$\rho^{(t',t)} = \prod_{l=1}^{L} \rho_l^{(t',t)}$	$\rho^{(t)} = \prod_{t'=1}^{t} \rho^{(t',t)}$	$\rho^{(\leq t)} = \prod_{t'=1}^{t} \rho^{(t')}$
FIM	$\mathbf{F}_l^{(t',t)}$	$\mathbf{F}^{(t',t)} = \text{diag}(\mathbf{F}_l^{(t',t)})_{l\in[L]}$	$\mathbf{F}^{(t)} = \text{diag}(\mathbf{F}^{(t',t)})_{t'\in[t]}$	$\mathbf{F}^{(\leq t)} = \text{diag}(\mathbf{F}^{(t')})_{t'\in[t]}$

Table 3.2.: The notation for PBNN. Here, $1 \leq t' \leq t$ and $l \in [L]$ for $t = t'$ and $l \in \mathfrak{L}$ for $t' < t$. For task 0, we write (0) instead of $(0, 0)$ as no inter-column weights are used.

column layers. Lastly, the superscript $(\leq t)$ denotes all layers up to column t. Table 3.2 summarizes the notation that is used for PBNN.

Prior. Because PBNN also use the fully Bayesian, frequentist, or combined approach described in Section 3.1, in this section, we mainly concentrate on the choice of the prior and the adaption of the equations for multiple columns when some weights are fixed. The goal of choosing the prior is on the one hand important to reduce the KL-divergence in the PAC-Bayesian bounds to achieve a better generalization and on the other hand to improve the posterior. An optimal prior should be broad enough such that the parameters can be optimized flexibly. Nonetheless, the prior should in the end also be close to the posterior. Hence, the parameters should not deviate too much from the ones from the prior.

Now consider that the columns for $1, \ldots, t-1$ are given. When a new task $\mathfrak{T}_t$ is observed, the distributions of all previous columns are fixed and a new column is appended.

Because all preceding columns are frozen and their posterior was already computed for their corresponding task, the prior is chosen to be their posterior from the earlier task. This means that we choose $\pi_l^{(i,j)} = \rho_l^{(i,j)}$ for all previous columns $i \leq j < t$ and all layers $l \in [L]$ if $i = j$ and the lateral connections $l \in \mathfrak{L}$ if $i < j$. For the main column consisting of all non-lateral layers, namely $\boldsymbol{\theta}^{(t,t)}$, we use the empirically learned prior which should correspond to a broad distribution around useful weights, *i.e.* $\pi_l^{(t,t)} = \mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(0)}, (\tilde{\mathbf{F}}_l^{(0)})^{-1})$. For the lateral connections, the prior is harder to choose as their feature embeddings are usually more specific than the ones from the empirically learned prior and should thus not be pushed to be close to the empirically learned prior distribution. Therefore, for a lateral connection $l \in \mathfrak{L}$ from column $t' < t$, we use the posterior of this layer as the prior for this lateral connection, *i.e.* $\pi_l^{(t',t)} = \rho_l^{(t',t')}$. One exception is the last layer as the dimension of the last weight matrix might differ. Hence, in the case of different output dimensions for different tasks, we use an isotropic Gaussian prior for the last layer $l = L$.

Summarized, this means that the prior for a layer $l \in [L]$ of the main column is given by

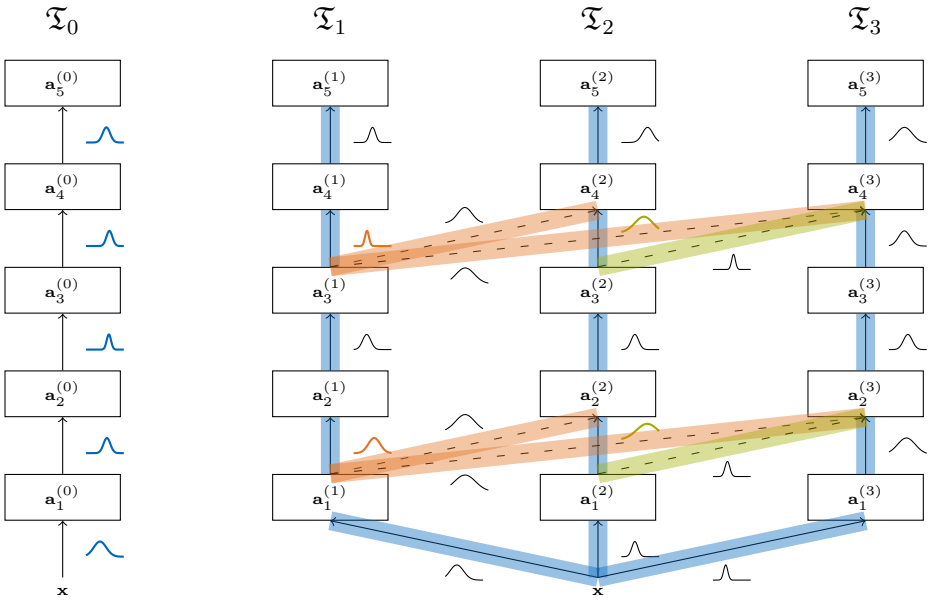


Figure 3.1.: The posterior is depicted by colored distributions. The colored shaded connection means that the posterior with the same color is the prior for this weight. A black distribution is not used as a prior in this setting. PBNN use the empirically learned prior from task $\mathfrak{T}_0$ (blue) in all main columns. The prior for the lateral connections is the posterior from the origin layer, where the lateral connection comes from (orange and green).

the empirical prior $\mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(0)}, (\tilde{\mathbf{F}}_l^{(0)})^{-1})$, where

$$\tilde{\mathbf{F}}_l^{(0)} = \beta_l^{(0)} \mathbf{F}_l^{(0)} + \alpha_l^{(0)} \gamma \mathbf{I}$$

and the prior for a lateral connection $l \in \mathfrak{L}$ leaving column t' is the posterior of column t', $\mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(t',t')}, (\tilde{\mathbf{F}}_l^{(t',t')})^{-1})$, where

$$\tilde{\mathbf{F}}_l^{(t',t')} = \beta_l^{(t',t')} \mathbf{F}_l^{(t',t')} + \alpha_l^{(t',t')} (\beta_l^{(0)} \mathbf{F}_l^{(0)} + \alpha_l^{(0)} \gamma \mathbf{I}).$$

All other weights are fixed and have the same prior and posterior. Figure 3.1 shows the structure of PBNN and the choice of the prior for each weight. Using this prior, we present the extension of the curvature scaling for PBNN. Similar to Section 3.1.3, we compute the approximate upper bound of the expected empirical error first and the KL-divergence for PBNN second.

Expected Empirical Error. To compute the approximate upper bound of the PAC-Bayesian bounds to optimize the curvature scales, one further needs to compute the expected empirical error on the training data as in Equation 3.8. Similar to Equation 3.6

in Section 3.1, one can compute the upper bound of the expected empirical error with

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{\theta}^{(\leq t)}\sim\rho^{(\leq t)}}\left[\frac{1}{N_t}\sum_{i=1}^{N_t}\mathrm{er}(f_{\boldsymbol{\theta}^{(\leq t)}},\mathbf{x}_i^{(t)},\mathbf{y}_i^{(t)})\right]\\
&\lessapprox \frac{1}{N_t\ln 2}\left(-\ln p(\mathcal{D}_t|f_{\hat{\boldsymbol{\theta}}^{(\leq t)}})+\frac{1}{2}\mathbb{E}_{\boldsymbol{\theta}^{(\leq t)}\sim\rho^{(\leq t)}}\left[(\boldsymbol{\theta}^{(\leq t)}-\hat{\boldsymbol{\theta}}^{(\leq t)})^T\mathbf{F}^{(\leq t)}(\boldsymbol{\theta}^{(\leq t)}-\hat{\boldsymbol{\theta}}^{(\leq t)})\right]\right),
\end{aligned}$$

where $\mathbf{F}^{(\leq t)}$ denotes the FIM on task t for all weights up to column t. The quadratic term in the expectation can be computed exactly by using that $\rho^{(\leq t)}$ is a normal distribution with mean $\hat{\boldsymbol{\theta}}^{(\leq t)}$. Furthermore, it can be decomposed into

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{\theta}^{(\leq t)}\sim\rho^{(\leq t)}}\left[(\boldsymbol{\theta}^{(\leq t)}-\hat{\boldsymbol{\theta}}^{(\leq t)})^T\mathbf{F}^{(\leq t)}(\boldsymbol{\theta}^{(\leq t)}-\hat{\boldsymbol{\theta}}^{(\leq t)})\right]=\mathrm{tr}(\mathbf{F}^{(\leq t)}\tau(\tilde{\mathbf{F}}^{(\leq t)})^{-1})\\
&=\sum_{j=1}^{t-1}\sum_{i=1}^{j}\mathrm{tr}(\mathbf{F}^{(i,j)}\tau(\tilde{\mathbf{F}}^{(i,j)})^{-1})+\sum_{i=1}^{t}\mathrm{tr}(\mathbf{F}^{(i,t)}\tau(\tilde{\mathbf{F}}^{(i,t)})^{-1}).
\end{aligned}$$

The weights from previous columns, *i.e.* for $i\leq j<t$, are frozen and their corresponding posterior together with their curvature scales are fixed. Therefore, the first term is constant with respect to the new curvature scales. To avoid computing the FIM with respect to previous network weights, we assume that $\mathbf{F}^{(i,j)}(\tilde{\mathbf{F}}^{(i,j)})^{-1}\approx\mathbf{I}$ for $i\leq j<t$. This simplifies the upper bound:

$$\begin{aligned}
&\sum_{j=1}^{t-1}\sum_{i=1}^{j}\mathrm{tr}(\mathbf{F}^{(i,j)}\tau(\tilde{\mathbf{F}}^{(i,j)})^{-1})+\sum_{i=1}^{t}\mathrm{tr}(\mathbf{F}^{(i,t)}\tau(\tilde{\mathbf{F}}^{(i,t)})^{-1})\\
&\quad\approx\tau\frac{(t-2)(t-1)}{2}\sum_{l\in\mathfrak{L}}n_t+(t-1)\sum_{l=1}^{L}n_t+\sum_{i=1}^{t}\mathrm{tr}(\mathbf{F}^{(i,t)}\tau(\tilde{\mathbf{F}}^{(i,t)})^{-1})\\
&\quad=:\tau n^{(\leq t-1)}+\sum_{i=1}^{t}\mathrm{tr}(\mathbf{F}^{(i,t)}\tau(\tilde{\mathbf{F}}^{(i,t)})^{-1})\\
&\quad=\tau n^{(\leq t-1)}+\sum_{i=1}^{t-1}\sum_{l\in\mathfrak{L}}\mathrm{tr}(\mathbf{F}_l^{(i,t)}\tau(\beta_l^{(i,t)}\mathbf{F}_l^{(i,t)}+\alpha_l^{(i,t)}\tilde{\mathbf{F}}_l^{(i,i)})^{-1})\\
&\qquad+\sum_{l=1}^{L}\mathrm{tr}(\mathbf{F}_l^{(t,t)}\tau(\beta_l^{(t,t)}\mathbf{F}_l^{(t,t)}+\alpha_l^{(t,t)}\tilde{\mathbf{F}}_l^{(0)})^{-1})\\
&\quad=\tau n^{(\leq t-1)}+\sum_{(i,l,\mathbf{z})\in\mathcal{I}}\sum_{\substack{\lambda\in\Lambda(A)\\ A=\mathbf{F}_l^{(i,t)}(\tilde{\mathbf{F}}_l^{\mathbf{z}})^{-1}}}\frac{\tau}{\beta_l^{(i,t)}+\alpha_l^{(i,t)}\lambda^{-1}}.
\end{aligned}\tag{3.13}$$

Here, we use again the index set $\mathcal{I}$ and introduce the number of parameters for all fixed previous columns as $n^{(\leq t-1)}$.

KL-Divergence. Assuming that weights of different layers are independent, the KL-divergence between the posterior and the prior can be written as

$$\begin{aligned}
\mathbb{KL}\left(\rho^{(\leq t)} \| \pi^{(\leq t)}\right) &= \sum_{j=1}^{t} \sum_{i=1}^{j} \mathbb{KL}\left(\rho^{(i,j)} \| \pi^{(i,j)}\right) \\
&= \sum_{i=1}^{t} \mathbb{KL}\left(\rho^{(i,t)} \| \pi^{(i,t)}\right) + \sum_{j=1}^{t-1} \sum_{i=1}^{j} \mathbb{KL}\left(\rho^{(i,j)} \| \pi^{(i,j)}\right) \\
&= \sum_{i=1}^{t} \mathbb{KL}\left(\rho^{(i,t)} \| \pi^{(i,t)}\right) \qquad (3.14) \\
&= \sum_{l \in [L]} \mathbb{KL}\left(\rho_l^{(t,t)} \| \mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(0)}, \tau(\tilde{\mathbf{F}}_l^{(0)})^{-1})\right) + \sum_{l \in \mathfrak{L}} \sum_{i=1}^{t-1} \mathbb{KL}\left(\rho_l^{(i,t)} \| \rho_l^{(i,i)}\right),
\end{aligned}$$

where it was used that the prior is equal to the posterior for all fixed columns in Equation 3.14. We approximate the posterior as in Section 3.1 with Laplace approximation and curvature scaling. Therefore, the posterior is also a normal distribution $\mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(i,t)}, \tau(\tilde{\mathbf{F}}_l^{(i,t)})^{-1})$, for lateral connections $l \in \mathfrak{L}$, $i < t$ and the main column $l \in [L]$, $i = t$, which results in a closed-form KL-divergence. Using the index set $\mathcal{I}$ including the lateral connections and the main column

$$\mathcal{I} = \{(i, l, (i, i)) \mid i \in [t-1],\ l \in \mathfrak{L}\} \cup \{(t, l, (0)) \mid l \in [L]\}$$

the KL-divergence can be written in terms of eigenvalues as in Section 3.1:

$$\begin{aligned}
\mathbb{KL}(\rho^{(\leq t)} \| \pi^{(\leq t)}) &= \sum_{l \in [L]} \mathbb{KL}\left(\mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(t,t)}, \tau(\tilde{\mathbf{F}}_l^{(t,t)})^{-1}) \| \mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(0)}, \tau(\tilde{\mathbf{F}}_l^{(0)})^{-1})\right) \\
&\quad + \sum_{l \in \mathfrak{L}} \sum_{i=1}^{t-1} \mathbb{KL}\left(\mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(i,t)}, \tau(\tilde{\mathbf{F}}_l^{(i,t)})^{-1}) \| \mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(i,i)}, \tau(\tilde{\mathbf{F}}_l^{(i,i)})^{-1})\right) \\
&= \frac{1}{2} \left(\sum_{(i,l,\mathbf{z}) \in \mathcal{I}} \frac{1}{\tau} (\hat{\boldsymbol{\theta}}_l^{(i,t)} - \hat{\boldsymbol{\theta}}_l^{\mathbf{z}})^T \tilde{\mathbf{F}}_l^{\mathbf{z}} (\hat{\boldsymbol{\theta}}_l^{(i,t)} - \hat{\boldsymbol{\theta}}_l^{\mathbf{z}}) + \sum_{\substack{\lambda \in \Lambda(A) \\ A = \mathbf{F}_l^{(i,t)} (\tilde{\mathbf{F}}_l^{\mathbf{z}})^{-1}}} \frac{1}{\beta_l^{(i,t)} \lambda + \alpha_l^{(i,t)}} - 1 - \ln(\beta_l^{(i,t)} \lambda + \alpha_l^{(i,t)}) \right). \qquad (3.15)
\end{aligned}$$

Altogether, this shows the advantage of PBNN that some related feature embeddings can increase the performance even though only the current column contributes to the KL-divergence.

Training of PBNN. Altogether, the training for column t given columns $1, \ldots, t-1$ consists of optimizing the parameters with the output of the t-th column either with MAP estimation or by optimizing a PAC-Bayesian bound on the dataset $\mathcal{D}_t$. During this optimization, we already sample the weights in the previous columns from their corresponding

posterior such that the current task can adapt to the incoming feature distributions. In a second step, the FIM with respect to the unfrozen weights $\boldsymbol{\theta}^{(t)}$, namely $\mathbf{F}^{(t)}$, is computed and used to determine the eigenvalues of the product between the FIM and the prior precision. The eigenvalues can then be used to estimate the KL-divergence and expected empirical error to optimize for the curvature scales for each layer. In a final step, the posterior precision matrix is computed as the linear combination of the prior precision and the FIM, weighted by the curvature scales and multiplied by temperature scaling as described in Section 3.1. The whole method is shown in Algorithm 1. Moreover, the further pseudo-code for the objective, curvature scaling, and progressive neural networks is given in Appendix A.2. After obtaining the optimal parameters and the precision matrix, one can then sample from the resulting normal distribution to exploit the Bayesian model averaging. It has to be noted that the order of datasets plays an important role in PBNN as later tasks can use the previous features but not the other way around.

Computational Complexity. For each task and additionally for the prior task, the computational complexity boils down to finding the optimal parameters, computing the FIM, determining the eigenvalues of the matrix product of the FIM with the prior covariance matrix, and scaling the curvature. The parameter optimization corresponds to finding the MAP estimate. This can be solved efficiently by computing the negative log-likelihood over mini-batches with common variations of stochastic gradient descent [54]. Additionally, in each update step, the quadratic term induced by the prior needs to be computed. As discussed in Section 3.1.3, for a diagonal or Kronecker-factored matrix, this term can be computed efficiently such that the parameter optimization has a similar complexity as standard MAP training. For common approximations of the FIM like the K-FAC and the K-FOC, the computation of the FIM corresponds to one additional epoch over the training data [38, 71]. Nonetheless, the K-FOC usually has a small linear overhead compared to K-FAC and hence, can also correspond to multiple epochs depending on the number of iterations in the K-FOC algorithm (see Appendix A.1.3). To compute the eigenvalues, one can again take advantage of the properties of the FIM as described in Section 3.1.3 which leads to a neglectable computational complexity in practice. The curvature scaling is an optimization problem with a total of $2L$ parameters, where L is the number of layers in the network. Hence, it is usually a much lower-dimensional optimization than finding the weights. Moreover, the computation of the negative log-likelihood can be incorporated in the computation of the FIM. Therefore, no additional pass through the data is needed. Altogether, during training, PBNN have a similar computational complexity as PNN with the main overhead of training one additional task. Nonetheless, this overhead can be reduced by using a pre-trained model which then leads to the minor overhead of about one additional epoch to compute the FIM for every task. During inference, PBNN use multiple network samples and thus, have a computational complexity comparable to multiple forward passes in PNN.

Extension for Arbitrary Network Architectures. So far, only a basic architecture without skip connections was considered. Nevertheless, PBNN can also be extended to arbitrary architectures including ResNets [43]. To achieve this generality, PBNN should

not depend on one specific forward pass implementation. Instead, the forward method should be managed by the network architecture that is given. In order to implement the lateral connections, PBNN save intermediate activations on the one hand and on the other hand introduce aggregation layers that combine the saved activations on the layer level. The first part is implemented by creating forward hooks for the lateral connection layers $l \in \mathfrak{L}$ that save the activations of the previous layer $a_{l-1}^{(t')}$ during the forward pass in the base network. To combine the activation, the aggregation layers apply the lateral layers on the saved activations of the previous columns, $\phi_l^{(t',t)}(a_{l-1}^{(t')})$, and compute the mean over the resulting pre-activations. The aggregation layer is included in the base network by exchanging each layer $l \in \mathfrak{L}$ by the composition of the layer followed by the aggregation layer. With this architectural change, PBNN can also be applied to complex network architectures.

Algorithm 1: Training of Progressive Bayesian Neural Networks.

input :

$f.$	network architecture
$(\mathcal{D}_t)_{t=0}^T$	datasets
$\mathfrak{L}$	lateral connections
γ	weight decay
τ	temperature scaling
$\varepsilon = 0.2$	PAC-Bayes probability

output : PBNN with posterior distribution $\mathcal{N}(\hat{\boldsymbol{\theta}}^{(\leq T)}, \tilde{\mathbf{F}}^{(\leq T)})$

function `fit_pbnn`$(f., (\mathcal{D}_t)_{t=0}^T, \mathfrak{L}, \gamma, \tau, \varepsilon)$

▷ Find optimal parameters $\hat{\boldsymbol{\theta}}^{(0)}$ for $\mathcal{D}_0$ (with MAP or PAC-Bayesian bounds)

$\hat{\boldsymbol{\theta}}^{(0)} \leftarrow \arg\min_{\boldsymbol{\theta}}$ `objective`$([L], f., \boldsymbol{\theta}, \mathcal{D}_0, (\mathbf{0})_{l\in[L]}, (\gamma\mathbf{I})_{l\in[L]}, \tau)$

▷ Compute a FIM approximation $\mathbf{F}^{(0)}$

$\mathbf{F}^{(0)} \leftarrow$ `FIM`$(f., \hat{\boldsymbol{\theta}}^{(0)}, \mathcal{D}_0)$

▷ Compute the eigenvalues for $\frac{1}{\gamma}\mathbf{F}_l^{(0)}$ for each layer $l \in [L]$

$\Lambda \leftarrow (\Lambda(\frac{1}{\gamma}\mathbf{F}_l^{(0)}))_{l\in[L]}$

▷ Find optimal curvature scales $\alpha_l^{(0)}$ and $\beta_l^{(0)}$ using Equation 3.15 and Equation 3.13

$nll \leftarrow -\ln(\mathcal{D}_0 | f_{\hat{\boldsymbol{\theta}}^{(0)}})$

$q \leftarrow \frac{\gamma}{2\tau}\|\hat{\boldsymbol{\theta}}^{(0)}\|_2^2$

$\alpha^{(0)}, \beta^{(0)} \leftarrow \arg\min_{\alpha,\beta}$ `curvature_scaling`$([L], ((\alpha_l, \beta_l))_{l\in[L]}, \Lambda, nll, q,$ `len`$(\mathcal{D}_0), \tau, 0, \varepsilon)$

▷ Compute the precision matrix

$\tilde{\mathbf{F}}_l^{(0)} \leftarrow \beta_l^{(0)}\mathbf{F}_l^{(0)} + \alpha_l^{(0)}\gamma I$ for $l \in [L]$

▷ Initialize the progressive neural network

$pnn \leftarrow$ `PNN`$(f., \mathfrak{L})$

$\hat{\boldsymbol{\theta}}^{(\leq 0)} \leftarrow \varnothing$

for $t \in [T]$ **do**

$\mathcal{I} \leftarrow \{(i, l, (i, i)) \mid i \in [t-1],\ l \in \mathfrak{L}\} \cup \{(t, l, (0)) \mid l \in [L]\}$

▷ Find optimal parameters $\hat{\boldsymbol{\theta}}^{(t)}$ for $\mathcal{D}_t$ using output t of pnn (with MAP or PAC-Bayesian bounds)

$\hat{\boldsymbol{\theta}}^{(t)} \leftarrow$
$\arg\min_{\boldsymbol{\theta}}$ `objective`$(\mathcal{I}, pnn^t_{(\boldsymbol{\theta}\sim\rho^{(\leq t-1)},\cdot)}, (\boldsymbol{\theta}_l^{(i,t)})_{(i,l,\_)\in\mathcal{I}}, \mathcal{D}_t, (\hat{\boldsymbol{\theta}}_l^{\mathbf{z}})_{(\_,l,\mathbf{z})\in\mathcal{I}}, (\tilde{\mathbf{F}}_l^{\mathbf{z}})_{(\_,l,\mathbf{z})\in\mathcal{I}}, \tau)$

$\hat{\boldsymbol{\theta}}^{(\leq t)} \leftarrow (\hat{\boldsymbol{\theta}}^{(\leq t-1)}, \hat{\boldsymbol{\theta}}^{(t)})$

▷ Compute a FIM approximation $\mathbf{F}^{(t)}$ using output t of pnn

$\mathbf{F}^{(t)} \leftarrow$ `FIM`$(pnn^t_{(\hat{\boldsymbol{\theta}}^{(\leq t-1)},\cdot)}, \hat{\boldsymbol{\theta}}^{(t)}, \mathcal{D}_t)$

▷ Compute the eigenvalues for $\mathbf{F}_l^{(i,t)}(\tilde{\mathbf{F}}_l^{\mathbf{z}})^{-1}$ for $(i, l, \mathbf{z}) \in \mathcal{I}$

$\Lambda \leftarrow (\Lambda(\mathbf{F}_l^{(i,t)}(\tilde{\mathbf{F}}_l^{\mathbf{z}})^{-1}))_{(i,l,\mathbf{z})\in\mathcal{I}}$

▷ Find optimal curvature scales $\alpha_l^{(i,t)}$ and $\beta_l^{(i,t)}$

$nll \leftarrow -\ln(\mathcal{D}_t | pnn^t_{\hat{\boldsymbol{\theta}}^{(\leq t)}})$

$q \leftarrow \frac{1}{2\tau}\sum_{(i,l,\mathbf{z})\in\mathcal{I}}(\hat{\boldsymbol{\theta}}_l^{(i,t)} - \hat{\boldsymbol{\theta}}_l^{\mathbf{z}})^T\tilde{\mathbf{F}}_l^{\mathbf{z}}(\hat{\boldsymbol{\theta}}_l^{(i,t)} - \hat{\boldsymbol{\theta}}_l^{\mathbf{z}})$

$\alpha^{(t)}, \beta^{(t)} \leftarrow$
$\arg\min_{\alpha,\beta}$ `curvature_scaling`$(\mathcal{I}, ((\alpha_l^{(i,t)}, \beta_l^{(i,t)}))_{(i,l,\mathbf{z})\in\mathcal{I}}, \Lambda, nll, q,$ `len`$(\mathcal{D}_t), \tau, n^{(\leq t-1)}, \varepsilon)$

▷ Compute the precision matrix and update the posterior

$\tilde{\mathbf{F}}_l^{(i,t)} \leftarrow \beta_l^{(i,t)}\mathbf{F}_l^{(i,t)} + \alpha_l^{(i,t)}\mathbf{F}_l^{\mathbf{z}}$ for $(i, l, \mathbf{z}) \in \mathcal{I}$

$\rho_l^{(i,t)} \leftarrow \mathcal{N}(\hat{\boldsymbol{\theta}}_l^{(i,t)}, (\tilde{\mathbf{F}}_l^{(i,t)})^{-1})$

4. Experiments and Results

This chapter validates the approaches empirically on the predictive performance, uncertainty, and generalization bounds. The respective evaluation metrics are introduced in the first section. In the second section, the empirically learned prior together with the curvature scaling is compared with deterministic pre-training using MAP and MLE, with MC dropout, and Bayesian neural networks (BNN) with isotropic Gaussian priors. This allows us to compare, on the one hand, our method with deterministic pre-training in terms of accuracy and, on the other hand, the uncertainty of the model with MC dropout and BNN with isotropic Gaussians. Furthermore, we can observe the quality of the posterior for large temperature scales and hence, the strength of the cold posterior effect [112] compared to isotropic Gaussian priors. The extension to continual learning problems is then shown in the third section with a small-scale and a large-scale experiment. Both experiments compare PBNN against standard PNN and PNN using MC dropout. Therefore, we have a deterministic and a Bayesian baseline for our method to again compare the accuracy and the uncertainty. Moreover, we utilize both K-FAC and K-FOC for all runs and compare both with the Diagonal approximation for all but the large-scale continual learning experiment to get to know the influence of the FIM approximation quality on the quality of the predictions. In addition, we also compare in every experiment the different approaches described in Section 3.1.3, *i.e.* the fully Bayesian approach and the frequentist and combined approach for both PAC-Bayesian bounds. Thus, we can also reason about the influence of scaling the curvatures with our approximate upper bound. We conclude this chapter with a short discussion of the results.

All experiments are implemented in PyTorch [84] on top of the curvature library [49, 64, 99]. Moreover, we use $B_{MC} = 100$ samples of the posterior to estimate the predictive distribution. The training of the parameters utilizes the Adam [54] variation of stochastic gradient descent with an automatic learning rate scheduler that halves the learning rate once the validation loss does not decrease for five consecutive epochs and stops the training if the validation loss does not decrease for ten consecutive epochs. To optimize the curvature scales, we also use the Adam optimizer with the learning rate 0.5 and an exponential decaying learning rate by $1 - 10^{-6}$ every step.

4.1. Evaluation Metrics

The predictive performance is evaluated with the accuracy on an independent test set and the uncertainty of the model is assessed by computing the out-of-distribution entropy on

a different dataset and the expected calibration error on the test set. Furthermore, to evaluate the separation of in-distribution and out-of-distribution samples, we consider a symmetric KL-divergence between the different entropy histograms. Moreover, the two generalization bounds from Section 2.4 are estimated.

Accuracy. The *accuracy* measures the percentage of correctly classified samples. It can be estimated over a dataset $\mathcal{D} = ((\mathbf{x}_i, \mathbf{y}_i))_{i=1}^N$ as

$$acc := \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[\arg\max_{\mathbf{y} \in \mathcal{Y}} p(\mathbf{y}|\mathbf{x}_i, f_{\boldsymbol{\theta}}) = \mathbf{y}_i] = 1 - \mathcal{L}_{\mathcal{D}}^{\text{er}}(f_{\boldsymbol{\theta}}).$$

Out-of-Distribution Entropy. *Entropy* measures the uncertainty of a discrete random variable X with the probability mass function p. In the extreme case that the random process is close to deterministic, the entropy is low, while it is maximal for a uniform distribution over the outcomes. It is defined as

$$H(X) := -\sum_{x \in \mathfrak{X}} p(x) \ln p(x)$$

It measures the expected information in *nats* over the distribution when the natural logarithm is used. In this thesis, the predictive uncertainty of the model on out-of-distribution samples is of special interest. For the out-of-distribution sample $(\mathbf{x}, \mathbf{y})$, it is defined as the entropy of the predictive distribution $p(\cdot|\mathbf{x}, f_{\boldsymbol{\theta}})$, *i.e.*

$$H(f_{\boldsymbol{\theta}}, \mathbf{x}) := -\sum_{\mathbf{y} \in \mathcal{Y}} p(\mathbf{y}|\mathbf{x}, f_{\boldsymbol{\theta}}) \ln p(\mathbf{y}|\mathbf{x}, f_{\boldsymbol{\theta}}).$$

To compute the out-of-distribution entropy on a dataset $\mathcal{D} = ((\mathbf{x}_i, \mathbf{y}_i))_{i=1}^N$, one usually estimates the expected predictive entropy over the samples:

$$h := \frac{1}{N} \sum_{i=1}^{N} H(f_{\boldsymbol{\theta}}, \mathbf{x}_i).$$

Altogether, this value should be high on out-of-distribution (OOD) data and low on in-distribution (ID) data.

Symmetric KL-Divergence. To assess how well the different models separate ID from OOD samples, we compute the entropy for each sample and then compare the distribution of the entropy values with histograms. Furthermore, the distance between these two distributions is commonly calculated with a symmetric version of the KL-divergence [69]. Given the probability mass functions p and q over a discrete space $\mathfrak{X}$ the KL-divergence [68] is similarly defined as in the continual setting but with a sum instead of an integral:

$$\mathbb{KL}(p||q) = \sum_{x \in \mathfrak{X}} p(x) \ln \left(\frac{p(x)}{q(x)} \right). \tag{4.1}$$

For the entropy histograms, the probabilities are the fraction of samples in the corresponding bin. Because the KL-divergence is not symmetric, Maddox *et al.* [69] use the sum of the KL-divergence in both directions:

$$\mathbb{KL}_{sym}(p, q) = \mathbb{KL}(p||q) + \mathbb{KL}(q||p). \tag{4.2}$$

However, the symmetric KL-divergence is infinity if exactly one probability distribution assigns no probability to a sample $x \in \mathfrak{X}$. Therefore, in practice, a small number, namely 10^{-7}, is added to both distributions after which the distributions are again normalized. The resulting symmetric KL-divergence is small if both distributions are similar and large if the distributions are separated.

Expected Calibration Error. Another measure for the uncertainty of a neural network is the calibration of its predictive probability. This reflects the belief that the confidence of the predictive distribution should be similar to the accuracy. Hence, the samples with a high certainty should have a high accuracy while samples that are predicted with a low accuracy should also have low confidence on these predictions. The *confidence* of a model for a data point $(\mathbf{x}, \mathbf{y})$ can formally be defined as the maximal probability of the output distribution, $\text{conf}(\mathbf{x}, f_{\boldsymbol{\theta}}) = \max_{\mathbf{y}' \in \mathcal{Y}} p(\mathbf{y}'|\mathbf{x}, f_{\boldsymbol{\theta}})$. The *expected calibration error* (ECE) is then the difference of the confidence and the accuracy computed over bins of samples. For this, the dataset is split into $q \in \mathbb{N}$ equally sized bins by their corresponding confidence. The bin B_j for $j \in [q]$ is defined as the set of samples

$$B_j = \left\{ (\mathbf{x}_i, \mathbf{y}_i) \mid \frac{j-1}{q} < \text{conf}(\mathbf{x}_i, f_{\boldsymbol{\theta}}) \leq \frac{j}{q} \right\}.$$

Therefore, the average accuracy and confidence on these bins can be computed for $j \in [q]$ as

$$acc_j = \frac{1}{|B_j|} \sum_{(\mathbf{x},\mathbf{y}) \in B_j} \mathbb{1}[\arg\max_{\mathbf{y}' \in \mathcal{Y}} p(\mathbf{y}'|\mathbf{x}, f_{\boldsymbol{\theta}}) = \mathbf{y}] \quad \text{and} \quad conf_j = \frac{1}{|B_j|} \sum_{(\mathbf{x},\mathbf{y}) \in B_j} \text{conf}(\mathbf{x}, f_{\boldsymbol{\theta}}),$$

respectively. The ECE is then the expected difference between the confidence and the accuracy in each bin weighted by the number of samples, *i.e.*

$$ECE := \sum_{j=1}^{q} \frac{|B_j|}{N} |acc_j - conf_j|.$$

This value should be small for a well-calibrated model.

PAC-Bayesian Bounds. In addition to the accuracy and uncertainty of the model, we are interested in the generalization bounds. For this, we consider the McAllester Bound (Equation 2.5) and the Catoni Bound (Equation 2.7). The expected empirical error is estimated jointly with the accuracy of the model using $B_{MC} = 100$ samples as

$$\mathbb{E}_{\boldsymbol{\theta} \sim \rho}[\mathcal{L}^{\text{er}}_{\mathcal{D}}(f_{\boldsymbol{\theta}})] \approx \frac{1}{B_{MC}} \sum_{i=1}^{B_{MC}} \mathcal{L}^{\text{er}}_{\mathcal{D}}(f_{\boldsymbol{\theta}_i}), \quad \boldsymbol{\theta}_i \sim \rho.$$

Dataset	Domain	Shape	Number of samples		
			Train	Val	Test
MNIST [61]	handwritten digits	$1\times28\times28$	$54,000$	$6,000$	$10,000$
NotMNIST[1]	roman letters	$1\times28\times28$	$54,000$	$6,000$	$10,000$
KMNIST [16]	Japanese letters	$1\times28\times28$	$54,000$	$6,000$	$10,000$
FashionMNIST [117]	fashion items	$1\times28\times28$	$54,000$	$6,000$	$10,000$
SVHN [80]	house numbers	$1\times32\times32$	$65,931$	$7,326$	$26,032$
CIFAR-10 [59]	real-world photos	$3\times32\times32$	$45,000$	$5,000$	$10,000$
ImageNet [20]	real-world photos	$3\times224\times224$	$1,281,167$	–	–
WRGBD [60]	every-day items	$3\times224\times224$	$111,184$	$27,924$	$68,812$

Table 4.1.: The datasets used in this thesis. In the small-scale experiments, the first six datasets are used. The last two datasets are used in the large-scale experiment. For this, the WRGBD dataset is further split into six different datasets.

As described in Section 3.2, the KL-divergence can be computed exactly with Equation 3.15. The bound of the probability in the PAC-Bayesian bound is chosen to be relatively broad with $\varepsilon = 0.8$ to make the expected empirical error and the KL-divergence the dominant terms in the PAC-Bayesian bounds.

4.2. Transfer Learning

The first experiment uses a LeNet-5 [62] architecture with two convolutional layers followed by three fully-connected layers. The empirically learned prior is evaluated in a transfer learning setting where the prior is learned over the MNIST dataset [61] and this prior is used to estimate the posterior on the NotMNIST dataset. The OOD entropy is computed over the test split of the FashionMNIST dataset [117] containing images of different fashion items. All datasets consist of gray-scale images of size 28×28. MNIST depicts handwritten digits, NotMNIST computer-written letters, and FashionMNIST fashion items like T-shirts and trousers. A summary of all datasets used in this thesis is given in Table 4.1. We compare the learned prior with an isotropic Gaussian prior with weight decay 10^{-3} and 10^{-5} in terms of accuracy and uncertainty. Both Bayesian neural networks are further compared to deterministic pre-trained MAP (with weight decay 10^{-3}, 10^{-5}, and 10^{-8}) and MLE training and to MC dropout (with dropout probability 0.01, 0.1, and 0.3 each without weight decay and with weight decay 10^{-5}). All models use the same pre-trained network either for deterministic pre-training or in the empirically learned prior. We do not use any data augmentation as we mainly focus on the method. To validate the curvature scaling, we compare the PAC-Bayesian bounds for all Bayesian neural networks for the differently learned curvature scales. As PAC-Bayesian bounds can only be computed for distributions over networks, the PAC-Bayesian bounds can not be

Table 4.2.: Results of the best models on the transfer learning experiment.

Method	FIM	γ	τ	*acc* (%)	OOD h	ID h	*ECE*
MLE	–	0.0	–	95.28	1.37	**0.11**	0.01
MAP	–	10^{-3}	–	95.33	1.44	0.13	0.01
MC dropout (probability 0.3)	–	0.0	–	**95.87**	1.65	0.21	0.02
BNN with isotropic Gaussian	–	10^{-3}	10^{-4}	95.35	1.64	0.21	0.02
	–	10^{-5}	10^{-3}	95.5	1.48	0.17	0.01
BNN with learned prior	Diagonal	10^{-8}	10^{-1}	95.09	**1.71**	0.26	0.03
	K-FAC	10^{-8}	10^{0}	94.97	1.60	0.23	0.02
	K-FOC	10^{-8}	10^{-3}	94.92	1.55	0.17	**0.01**

used for MAP and MLE training as their distributions are null sets with respect to the non-degenerate continuous prior and therefore their KL-divergence would be infinite. The same also holds for the MC dropout as the set of possible networks is again finite. For the Bayesian neural networks, we use the temperature scalings $10^{-5}, \dots, 10^{0}$. All experiments are run with an Nvidia RTX 3060 GPU with 12 GB of GPU memory, batch size 256, and learning rate $5 \cdot 10^{-4}$.

Results. In the following, we first validate the empirically learned prior against deterministic and Bayesian baselines in terms of accuracy, OOD entropy, and calibration. Moreover, the prior is compared to isotropic Gaussians to determine how well this prior is specified. Secondly, the different curvature scaling approaches are evaluated using PAC-Bayesian bounds.

Empirical Prior Learning. Table 4.2 shows the best model with respect to the accuracy on the test dataset for each method. Moreover, for the Bayesian neural networks, the temperature scaling τ is shown. One can see that all models achieve a similar accuracy with a slight advantage for MC dropout. The OOD entropy h is in general higher for the Bayesian models, in particular for the isotropic Gaussian prior with more weight decay, MC dropout, and the marginally superior empirically learned prior with the diagonal FIM approximation. The expected calibration error *ECE* is low for all models even though the BNN with the learned prior is slightly better than the other methods. Overall, there is no clear dominance of a specific model in this setting. Furthermore, all three curvature approximation types – diagonal, K-FAC, and K-FOC – produce similar results and have models with around 95% accuracy.

Moreover, we follow Wenzel *et al.* [112] to test whether the empirically learned prior is better specified than an isotropic Gaussian prior. For this, we analyze the accuracy of the

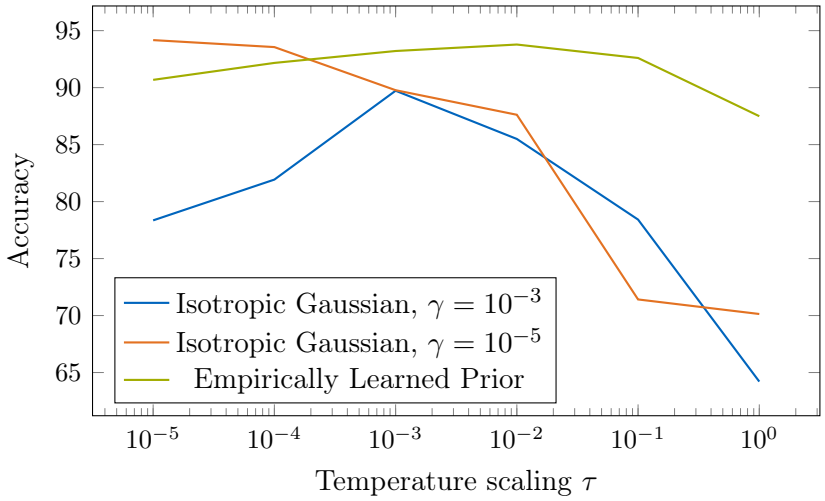


Figure 4.1.: The empirically learned prior is compared with isotropic Gaussian priors with weight decay $\gamma = 10^{-3}$ and $\gamma = 10^{-5}$ on the average accuracy for different temperature scalings. For large temperature scalings, it surpasses the other two models which is an indication that it is a better specified prior.

methods for different temperature scalings. The average value over all models for each prior is shown in Figure 4.1. We can see a performance drop for temperature scalings close to one for all models. Nonetheless, the empirically learned prior surpasses the other priors for large temperature scalings and achieves the best result for a temperature scaling of 10^{-1}. This supports the claim that empirically learned priors are better specified than isotropic Gaussians and even though they do not solve the cold posterior problem completely, they improve the results for larger temperature scalings.

Curvature Scaling. The curvature scaling is validated on the PAC-Bayesian bounds. Table 4.3 shows the PAC-Bayesian bounds for the best model with respect to the Catoni bound for each approach. While the scaling with the Catoni bound does not lead to any noticeable improvements, using the McAllester bound either only for the curvature scaling or also for parameter optimization produces non-vacuous generalization bounds. In detail, the error bounds are reduced by 19 and 23 percent points for the combined and frequentist approach, respectively, compared to the fully Bayesian approach. On top of that, the OOD entropy and accuracy are also similar. The poor results of Catoni scaling could be due to optimization difficulties because of the exponential in the bound. Therefore, it is often difficult to choose suitable learning rates for the stochastic gradient descent variations. Since the optimization of the curvature scales is in a relatively low-dimensional space compared to the parameter space, future work could also consider using other optimization methods to find optimal curvature scales with respect to the Catoni bound.

Another interesting finding is that all optimal models with respect to the Catoni bound use the empirically learned prior instead of the isotropic Gaussian. Altogether, empirically learned priors have on average a Catoni bound of 0.90 while both isotropic Gaussian priors

have an average bound of 0.99. A possible reason for this is that the posterior can be closer to the prior as many parameters can directly be used which in return leads to a lower KL-divergence and hence to lower PAC-Bayesian bounds.

Table 4.3.: PAC-Bayesian bounds for different approaches on the transfer learning experiment.

		PAC-Bayes bounds				
Approach	Curvature Scaling	Catoni	McAllester	*acc* (%)	OOD h	ID h
Fully Bayesian	$\alpha = 1$, $\beta = 1$	0.88	1.00	93.11	1.52	0.28
Combined	McAllester	0.69	0.71	88.90	1.71	0.56
	Catoni	0.88	1.00	93.11	**1.80**	0.59
Frequentist	McAllester	**0.65**	**0.68**	93.79	1.79	0.32
	Catoni	0.79	0.85	**94.64**	1.62	**0.27**

4.3. Continual Learning

The PBNN are validated using the continual learning setting. Here, the PBNN are compared with PNN using weight decay $10^{-1}, 10^{-2}, \ldots, 10^{-6}$ and PNN using MC dropout with dropout probabilities 0.01, 0.1, and 0.3 each with weight decay 10^{-5} and without weight decay. The temperature scaling is chosen dependent on the run using the first dataset to have a narrow enough posterior such that the samples produce usable predictions. The networks are evaluated on their accuracy and expected calibration error on each task. Moreover, the OOD entropy is computed for the test splits of the other tasks.

4.3.1. Small-Scale Experiment

Experimental Setup. For the small-scale experiment, we follow Ritter *et al.* [93]. Hence, the relevant tasks $\mathfrak{T}_1, \ldots, \mathfrak{T}_5$ are MNIST [61], NotMNIST, FashionMNIST [117], SVHN [80], and CIFAR-10 [59], respectively. Additionally, the prior is computed over KMNIST [16]. While KMNIST, MNIST, and NotMNIST are black-and-white images, FashionMNIST consists of gray-scale images and SVHN and CIFAR-10 are datasets with RGB images. This introduces the possibility to evaluate PBNN on different input distributions on the one hand. On the other hand, the domains are also different as KMNIST, MNIST, NotMNIST, and SVHN show digits and letters, FashionMNIST contains images of fashion items, and CIFAR-10 images of everyday objects. Because of the different shapes of the images, we repeat the first channel for the gray-scale and black-and-white images and pad all images to be of size $3 \times 32 \times 32$. The network is a LeNet-5 [62] as in the transfer learning experiment but with three input channels in the first layer. We use lateral connections for each parameterized layer except the first and the last layer. As all tasks have the

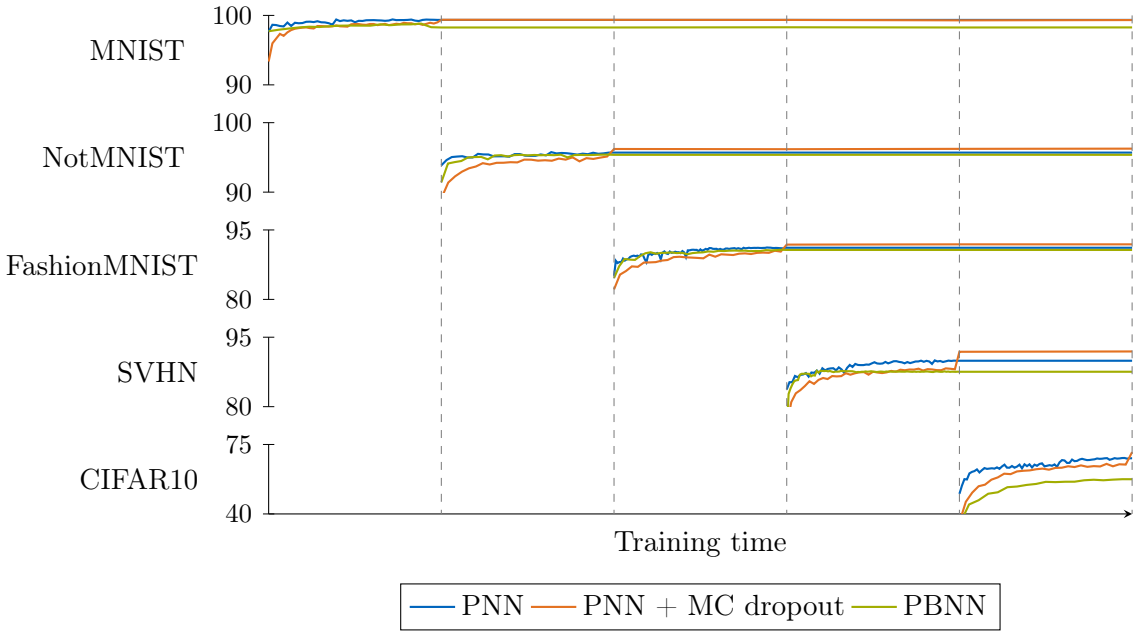


Figure 4.2.: The accuracy over the training time for the different tasks of the small-scale continual learning experiment is depicted for standard progressive neural networks (PNN), in combination with MC dropout (PNN + MC dropout) and progressive Bayesian neural networks (PBNN). During the training of each task, only one model sample is evaluated while after the weights are frozen, $B_{MC} = 100$ samples are used. All models have a similar accuracy on MNIST, NotMNIST, and FashionMNIST. For SVHN and CIFAR-10, the accuracy drops by a large margin for PBNN. Furthermore, model averaging improves the accuracy for MC dropout while it slightly worsens the performance of PBNN.

same output dimension, we use the empirically learned prior also on the last layer. The training was performed on an Nvidia GeForce GTX Titan X GPU with 12 GB of GPU memory, batch size 256, and learning rate $2 \cdot 10^{-3}$.

Results. Here, we first discuss the accuracy and uncertainty for PBNN and continue with an analysis of the generalization bounds.

Accuracy and Uncertainty. Because we aim to solve a task-incremental continual learning problem, we first look at the ability of the model to learn multiple sequentially incoming tasks. Figure 4.2 shows the accuracy on the test data of the different tasks over the training time for each method. It should be noted that during the training only one sample is used to evaluate the accuracy and the model averaging is only utilized when the training on the corresponding task is finished. The test accuracy for each task is further shown in Table 4.4. As expected, all models are able to learn the tasks sequentially without catastrophic forgetting. PNN with MC dropout consistently outperform

Table 4.4.: Results of the best models of the small-scale continual learning experiment.

		Tasks					
Method	Metric	MNIST	NotMNIST	FashionMNIST	SVHN	CIFAR-10	Average
PNN	*acc* (%)	99.35	95.69	91.15	89.92	68.05	88.83
	OOD h	1.37	1.71	**1.14**	0.55	0.85	1.13
	ID h	0.02	0.13	0.25	0.38	**0.79**	**0.31**
	ECE	**0.00**	**0.00**	**0.00**	**0.02**	0.04	**0.01**
PNN + MC dropout	*acc* (%)	**99.37**	**96.23**	**91.88**	**91.92**	**71.20**	**90.12**
	OOD h	**1.49**	**1.84**	1.06	**1.09**	**1.21**	**1.34**
	ID h	0.04	0.19	0.25	0.39	0.92	0.36
	ECE	0.01	0.02	0.01	0.04	0.04	0.02
PBNN	*acc* (%)	98.31	95.36	90.70	87.55	57.50	85.88
	OOD h	1.16	1.56	0.39	0.38	0.83	0.86
	ID h	**0.02**	**0.09**	**0.15**	**0.25**	1.07	0.32
	ECE	0.01	0.02	0.03	0.04	**0.04**	0.03

the other methods. Especially the model averaging improves the accuracy for all tasks. PBNN perform comparably for the first three tasks only lacking around one percent point compared to the best model. For the last two tasks, the accuracy is about four and twelve percent points worse than PNN with MC dropout, respectively. Hence, the domain switch from letters and digits to fashion items is not as problematic for PBNN as the change of gray-scale to RGB images. In this experiment, the K-FOC produces the most accurate results averaged over all tasks with an accuracy of 85.87% followed by K-FAC with 80.74% and the diagonal approximation with 75.52%.

Table 4.4 further shows the entropy for ID and OOD samples and the expected calibration error for each task. One can see that PNN are slightly better calibrated than the other two methods. Still, the uncertainty estimates of PNN together with MC dropout are for most tasks better. For this, the OOD entropy for each individual task and dataset is shown in Figure 4.3. PNN with MC dropout have the best uncertainty estimates over all columns, while PNN only have low OOD entropy in the fourth column and PBNN have a high OOD entropy for the first two columns and a low OOD compared to the ID entropy for the last three columns. Hence, PNN with dropout leads to the best separation between ID and OOD samples. This is further discussed in Appendix A.3.1.

Overall, the empirically learned prior can impair the performance if the data distribution for the prior learning is different. In particular, the performance drop with respect to accuracy and uncertainty is the largest for CIFAR-10, which has a different domain and a different distribution over the input space than the KMNIST which was used to learn the prior. Nonetheless, already switching the domain with FashionMNIST or the input distribution with SVHN can produce worse results. This emphasizes the importance to find a suitable dataset to learn the prior.

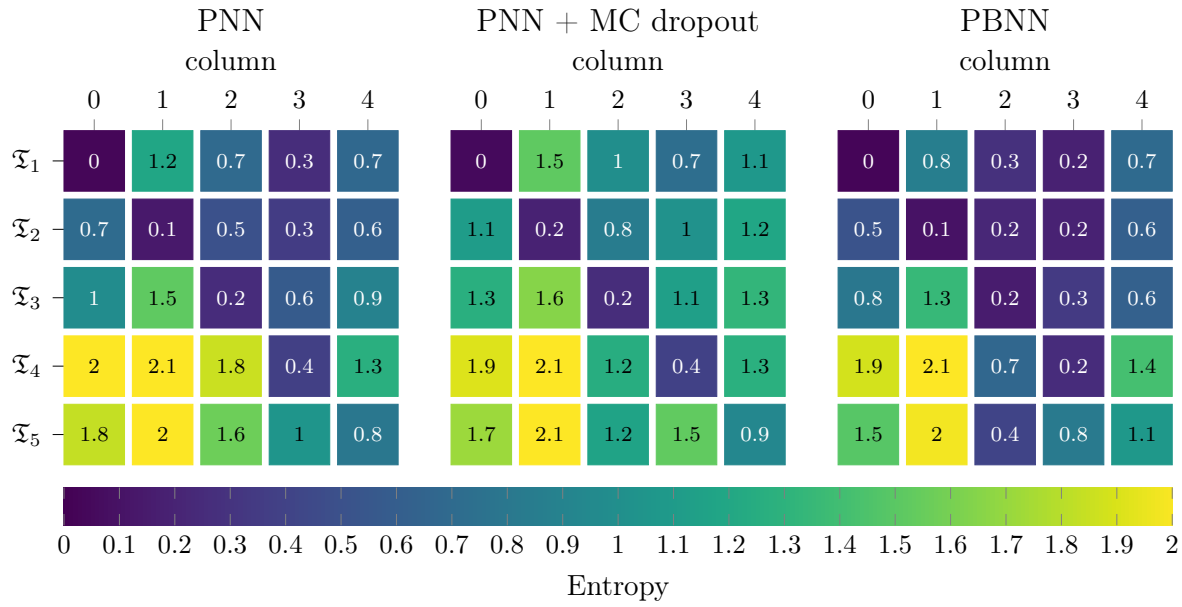


Figure 4.3.: The entropy for the different models is shown for each column and task. A good model has low entropy on the diagonal and high entropy on the off-diagonal elements. The OOD entropy is highest for PNN with MC dropout followed by PNN. PBNN have good uncertainty estimates for the first two columns but have low OOD entropy for the later columns. For PNN, the OOD entropy is also in general high. Still, the fourth output has low entropy on the first tasks. The entropy histograms are given in Figure A.3, Figure A.4, and Figure A.5 in Appendix A.3.1.

Generalization Bounds. For all approaches, the generalization bounds are only non-vacuous on the first column for the MNIST dataset. This has mainly two reasons: On the one hand, the first column has no lateral connections that increase the number of parameters. Therefore, the KL-divergence is usually smaller. On the other hand, the empirical error on the training data is the lowest for MNIST which directly influences the generalization bounds. Hence, in the following, we compare the generalization bounds on the MNIST dataset.

Table 4.5.: PAC-Bayesian bounds for different approaches on the small-scale continual learning experiment for MNIST.

		PAC-Bayes bounds	
Approach	Curvature Scaling	Catoni	McAllester
Fully Bayesian	$\alpha = 1$, $\beta = 1$	**0.53**	**0.59**
Combined	McAllester	0.87	0.96
	Catoni	0.97	1.34
Frequentist	McAllester	0.99	1.45
	Catoni	0.99	1.49

Table 4.5 shows the generalization bounds for the different approaches. In this experiment, both frequentist methods fail and optimizing the Catoni bound leads to loose bounds. The combined approach with the McAllester curvature scaling leads to better bounds but is outperformed by the fully Bayesian approach. This shows the current limitation of the curvature scaling that it depends on the FIM computed at the mode. Hence, if the FIM is very different from the precision matrix of the prior, it is possible that scaling both factors is not enough to achieve low generalization bounds. Nevertheless, further work could concentrate on finding modes that have a good curvature for instance by computing a batch estimate of the FIM after every epoch in combination with the herein derived approximate generalization bounds. Another option would be to directly optimize the bound for the optimal posterior precision matrix instead of for the curvature scales which would then lead to a frequentist method to obtain a normal distribution around that mode that has tighter generalization bounds.

4.3.2. Large-Scale Experiment

Experimental Setup. The large-scale continual learning experiment uses ImageNet (ILSVRC-2012) [20] to learn the prior and a ResNet-50 [43] architecture. This network uses residual connections and batch normalization to improve the training with deep neural networks. Following Grosse & Martens [38], we do not set a prior on the parameters of batch normalization. We specify a total of four lateral connections, each at the downsampling 1×1 convolution of the first "bottleneck" building block of the ResNet-layers. The relevant tasks are from the Washington RGB-D Object dataset [60] (WRGBD) and we

Table 4.6.: Results of the best models on the large-scale continual learning experiment.

		Tasks						
Method	Metric	Other	Banana	Coffee Mug	Stapler	Flashlight	Apple	Average
PNN	*acc* (%)	96.49	99.18	**100.00**	92.21	98.92	92.63	96.57
	OOD h	**2.41**	0.34	**0.65**	**0.65**	**0.34**	0.57	**0.83**
	ID h	0.06	0.08	0.01	0.07	0.05	0.17	0.07
	ECE	**0.02**	0.02	0.00	0.05	0.01	0.03	0.02
PNN + MC dropout	*acc* (%)	96.46	98.37	**100.00**	92.68	99.78	97.51	97.46
	OOD h	1.36	0.25	0.56	0.40	0.21	**0.60**	0.56
	ID h	0.04	0.14	**0.00**	0.08	0.03	0.06	0.06
	ECE	0.02	0.03	**0.00**	0.04	0.01	0.01	0.02
PBNN	*acc* (%)	**97.01**	**100.00**	**100.00**	**95.18**	**100.00**	**99.90**	**98.68**
	OOD h	0.75	**0.52**	0.61	0.50	0.30	0.39	0.51
	ID h	**0.03**	**0.03**	0.01	**0.04**	**0.00**	**0.03**	**0.02**
	ECE	0.02	**0.01**	0.00	**0.03**	**0.00**	**0.01**	**0.01**

closely follow the setting by Denninger & Triebel [21]. The dataset is a robotics dataset consisting of 51 object classes with a total of 300 different instances and over 160,000 images. Pictures of each instance were taken from three different angles – 30°, 45°, and 60°. The images from 45° are used as a test set and every fifth image from 30° and 60° is in the validation set. The remaining images are used for training. Furthermore, we only use the RGB channels and neglect the depth information. Following Denninger & Triebel [21], we also do instance recognition in a continual learning setting by splitting the classes banana, coffee mug, stapler, flashlight, and apple from the remaining classes. Task $\mathfrak{T}_1$ then consists of all remaining classes while task $\mathfrak{T}_2$ includes the bananas, task $\mathfrak{T}_3$ the coffee mugs, task $\mathfrak{T}_4$ the staplers, task $\mathfrak{T}_5$ the flashlights, and task $\mathfrak{T}_6$ the apples. For each task, the different instances should be predicted. All images are normalized by the mean and standard deviation from the ImageNet dataset and resized to the size 224×224. The experiment was run on an Nvidia RTX 3090 GPU with 24 GB of GPU memory, batch size 4, and learning rate 10^{-4}.

Results. The results of the large-scale continual learning experiment are shown in Table 4.6. Here, the best models for PNN, PNN with MC dropout and PBNN with respect to accuracy, ID and OOD entropy, and expected calibration error are compared on all tasks. All models are able to sequentially learn all tasks and PBNN are on average about one percent point more accurate than the other approaches. The continual learning plot is given in Figure 4.4. One can see that the training of PBNN is more stable and improves the accuracy on each dataset compared to the other two methods. In particular, for the stapler dataset, the accuracy is around three percent points higher. While PNN are better on the first two datasets, MC dropout enhances the predictions on the last three datasets. Moreover, all PBNN models apart from the ones using McAllester scaling produce consistently better results than PNN and their variation with MC dropout

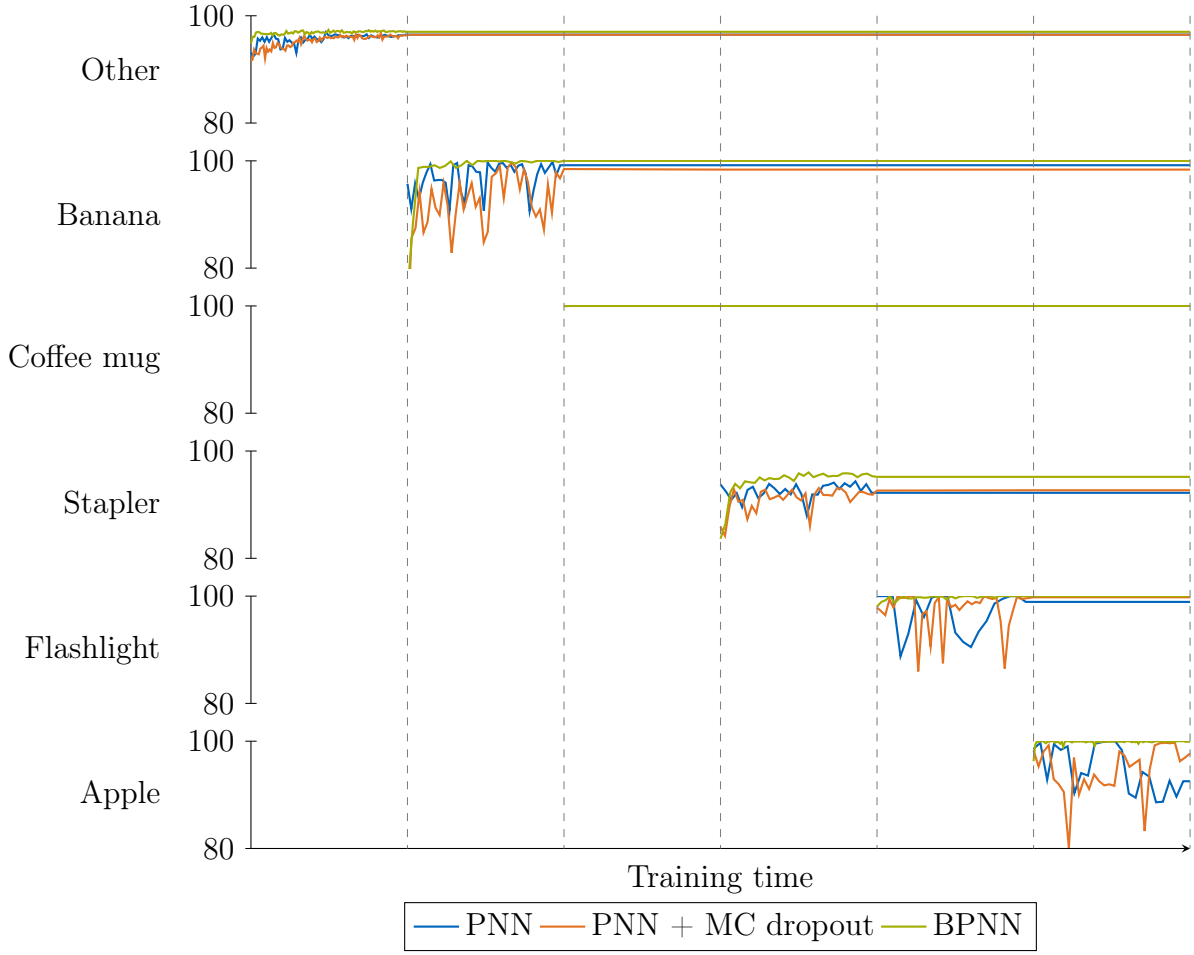


Figure 4.4.: The accuracy over the training time for the different tasks of the large-scale continual learning experiment is depicted for standard progressive neural networks (PNN), in combination with MC dropout (PNN + MC dropout) and progressive Bayesian neural networks (PBNN). All models mitigate catastrophic forgetting and PBNN produce consistently more accurate predictions than the other two methods.

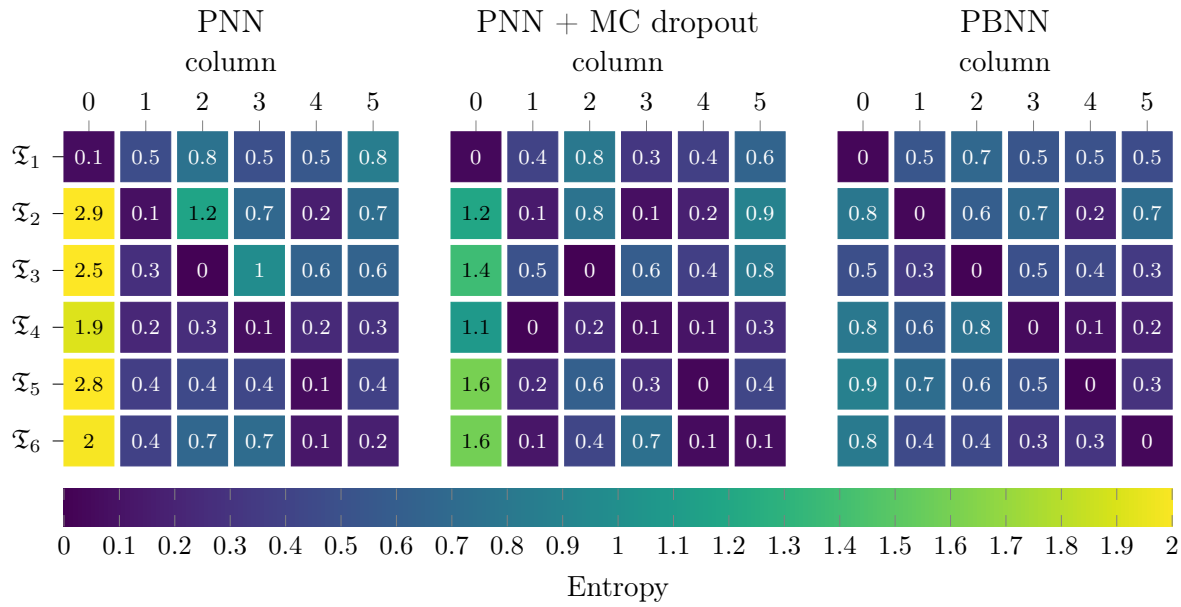


Figure 4.5.: The entropy for the different models is shown for each column and subsequent task. A good model has low entropy on the diagonal and high entropy on the off-diagonal elements. PNN and PNN with MC dropout have high OOD entropy for the first column. Nonetheless, the other columns have much lower OOD entropy. In contrast, PBNN have a similarly high OOD entropy for all columns. The entropy histograms are given in Figure A.6, Figure A.7, and Figure A.8 in Appendix A.3.2.

independent of whether K-FAC or K-FOC and which approach is used. K-FAC performs about 0.3 percent points better than the corresponding K-FOC model. The expected calibration error is also minimal for PBNN. Because the network is a lot larger than LeNet-5, the generalization bounds are vacuous for all tasks. Even though PNN have the highest OOD entropy among the three methods, Figure 4.5 shows that this term is dominated by the large OOD entropy for the first column. The other columns have worse uncertainty estimates. PNN with MC dropout have the same problem, but with an overall lower OOD entropy. Both methods have for instance a lower entropy on the output of column five for task six than the corresponding column. In practice, this means that, given an image of a banana, the column trained on flashlight images is more certain that the image shows a flashlight instance than the column trained on banana images is that it shows a banana instance. This problem does not appear in PBNN and even though the average OOD entropy is lower for PBNN, it is well distributed and each column has higher uncertainty on different datasets than it has on its corresponding test data. The same qualitative behavior can also be seen when the entropy histograms (Appendix A.3.2) are considered. Here, unlike PBNN, both PNN models are not able to separate the ID from the OOD samples based on the entropy. Altogether, this experiment shows that PBNN are scalable to large datasets and even outperform current state-of-the-art methods in terms of accuracy and uncertainty estimation.

4.4. Discussion

In total, we have conducted three experiments in this thesis – one transfer and two continual learning experiments.

From the transfer learning experiment, one can conclude that even though the empirically learned prior leads to no large improvements in terms of accuracy and uncertainty, it needs less temperature scaling for comparable results. Hence, the cold temperature effect is reduced but still not completely mitigated by learning the prior from a related dataset. In addition, curvature scaling is able to reduce PAC-Bayesian bounds and, when combined with the learned prior, leads to non-vacuous generalization bounds. Still, the improvements by curvature scaling depend on the similarity between the FIM and the precision matrix of the prior as shown in the small-scale continual learning experiment. Altogether, the empirical prior learning together with the curvature scaling introduces a novel method that learns the prior on a related dataset rather than directly specifying the complex prior in the weight space.

In the large-scale continual learning experiment, one can observe that PBNN improve the accuracy as well as the uncertainty compared to standard PNN and PNN with MC dropout. Moreover, the prior is learned over the large-scale ImageNet dataset and a ResNet-50 is used as a base network which shows the scalability of the method. However, the small-scale continual learning experiment shows that an unrelated prior can impair the performance of PBNN. Nevertheless, for many practical applications, related datasets are available that can be used to learn a suitable prior. As PBNN lead to well-separated entropy histograms between the ID and OOD entropy, the entropy of the columns could further be used to extend PBNN to the class-incremental setting, where the task id is not known during inference by using the most certain column.

5. Conclusion and Outlook

5.1. Conclusion

This thesis introduces a scalable method to utilize related datasets in order to improve generalization bounds and uncertainty estimates. In particular, it proposes to learn the prior distribution of deep Bayesian neural networks from large-scale datasets. This transforms the problem of finding a well-specified high-dimensional prior to searching related datasets to learn the prior. In this context, existing PAC-Bayesian bounds are used to derive a novel approximate upper bound of the generalization error for Laplace approximation which is then used to automatically scale the curvature. Empirically, this method obtains non-vacuous generalization bounds on NotMNIST and improves the generalization bounds by a large margin compared to standard Laplace approximation with isotropic Gaussians.

Furthermore, Progressive Bayesian Neural Networks (PBNN) are developed which is an extension of empirically learned priors to the continual learning setting, where previously learned feature embeddings and distributions are used to improve subsequent tasks. This builds on top of progressive neural networks and uses a well-informed prior distribution over the weights to generate usable uncertainty estimates. This is empirically validated on a large robotics dataset where it improves accuracy and uncertainty estimates compared to standard progressive neural networks. In addition, the Kronecker-Factored Optimal Curvature (K-FOC) is proposed in order to approximate the Fisher Information Matrix (FIM) more accurately, while preserving the Kronecker structure. For this, we perform experiments to demonstrate that K-FOC is closer to the FIM than the widely adopted K-FAC method.

5.2. Outlook

Even though PBNN can already be utilized to generate uncertainty estimates and reduce generalization bounds, there are some straightforward ways to improve them. Given that one already computes the FIM for PBNN, one could easily incorporate this information to prune the network architecture and to remove unnecessary weights [76, 120]. In particular, the model is pushed to use all lateral connections in PBNN as the prior might put few probability mass around zero. Hence, pruning lateral connections could on the one hand lead to a higher accuracy as useless connections can be ignored by the model. On the other

hand, pruning weights further reduces the number of weight and thus the KL-divergence, leading to better PAC-Bayesian bounds if high accuracy can be preserved [121].

One drawback of PBNN is that the size of the network increases linearly and the number of lateral connections even grows quadratic with the number of tasks. Therefore, one option would be to use one common backbone among all tasks and only utilize the progressive structure in the last few layers. Furthermore, this backbone could be the pre-trained neural network from the prior such that only the last layers contribute to the KL-divergence. This could further reduce the KL-divergence and recent work shows that having few or even only one Bayesian layer in a neural network can be enough to generate usable uncertainty estimates [58].

Another limiting factor of PBNN is the manual scaling of the temperature scaling. As the prior should not depend on the training data set, it is in general not possible to scale the temperature scaling using the PAC-Bayesian bounds. Nevertheless, it is possible to use a prior that is dependent on the training data distribution without depending too much on each sample. Hence, it might be possible to optimize the temperature scaling using PAC-Bayesian bounds when it can be ensured that it is not too dependent on each data sample, *e.g.* with differential privacy [26] or stability conditions [94].

Overall, we only use two basic PAC-Bayesian bounds in this thesis. Nonetheless, tighter and more general bounds are available [107, 109]. Because we derived the KL-divergence and the expected empirical error for PBNN, also other PAC-Bayesian bounds can be used and these two terms can just be plugged in to obtain a new approximate generalization bound for Laplace approximation that can directly be optimized with respect to the curvature.

Additionally, in this thesis, our empirically learned prior is only compared to isotropic Gaussian priors. However, an interesting direction would be to compare this prior with more advanced priors such as functional priors. In particular, the question under which circumstances each prior excels and which prior should be chosen for a given use-case should be addressed in the future.

Bibliography

1. Anderson, A., Shaffer, K., Yankov, A., Corley, C. D. & Hodas, N. O. Beyond fine tuning: A modular approach to learning on small data. *arXiv preprint arXiv:1611.01714* (2016).
2. Ba, J., Grosse, R. B. & Martens, J. *Distributed Second-Order Optimization using Kronecker-Factored Approximations* in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings* (2017).
3. Barber, D. & Bishop, C. M. Ensemble learning in Bayesian neural networks. *Generalization in Neural Networks and Machine Learning,* 215–237. `https://www.research.ed.ac.uk/en/publications/ensemble-learning-in-bayesian-neural-networks` (1998).
4. Becker, S. & Lecun, Y. Improving the convergence of back-propagation learning with second-order methods. *Proceedings of the 1988 Connectionist Models Summer School, San Mateo,* 29–37. `https://nyuscholars.nyu.edu/en/publications/improving-the-convergence-of-back-propagation-learning-with-secon` (1989).
5. Bindel, D. *Power Iteration* 2016. `https://www.cs.cornell.edu/%C2%A0bindel/class/cs6210-f16/lec/2016-10-17.pdf`.
6. Bishop, C. M. Bayesian Neural Networks. *Journal of the Brazilian Computer Society* **4.** ISSN: 0104-6500 (1997).
7. Bishop, C. M. *Pattern recognition and machine learning* Corrected at 8th printing 2009. ISBN: 0387310738. `http://swbplus.bsz-bw.de/bsz370363388cov.htm` (Springer, New York, NY, 2009).
8. Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association* **112,** 859–877 (2017).
9. Blum, A., Hopcroft, J. & Kannan, R. in *Foundations of Data Science* 29–61 (Cambridge University Press, 2020).
10. Botev, A., Ritter, H. & Barber, D. *Practical Gauss-Newton Optimisation for Deep Learning* in *Proceedings of the 34th International Conference on Machine Learning* (eds Precup, D. & Teh, Y. W.) **70** (PMLR, 2017), 557–565. `https://proceedings.mlr.press/v70/botev17a.html`.
11. Brea, J., Simsek, B., Illing, B. & Gerstner, W. *Weight-space symmetry in deep networks gives rise to permutation saddles, connected by equal-loss valleys across the loss landscape* 2019. `http://arxiv.org/pdf/1907.02911v1`.
12. Brown, T. *et al. Language Models are Few-Shot Learners* in *Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F. & Lin, H.) **33** (Curran Associates, Inc, 2020), 1877–1901. `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.
13. Catoni, O. *Pac-Bayesian supervised classification: The thermodynamics of statistical learning* ISBN: 0940600722 (Inst. of Math. Statistics, Beachwood, Ohio, 2007).

14. Chen, T., Fox, E. & Guestrin, C. *Stochastic Gradient Hamiltonian Monte Carlo* in *Proceedings of the 31st International Conference on Machine Learning* (eds Xing, E. P. & Jebara, T.) **32** (PMLR, Bejing, China, 2014), 1683–1691. https://proceedings.mlr.press/v32/cheni14.html.

15. Chen, Z. & Liu, B. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* **12,** 1–207 (2018).

16. Clanuwat, T. *et al. Deep Learning for Classical Japanese Literature* 2018.

17. Clark, E. *et al. All That's 'Human'Is Not Gold: Evaluating Human Evaluation of Generated Text* in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (2021), 7282–7296.

18. Cordts, M. *et al. The Cityscapes Dataset for Semantic Urban Scene Understanding* in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

19. Delange, M. *et al.* A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

20. Deng, J. *et al. ImageNet: A large-scale hierarchical image database* in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, Piscataway, NJ, 2009), 248–255. ISBN: 978-1-4244-3992-8.

21. Denninger, M. & Triebel, R. *Persistent anytime learning of objects from unseen classes* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), 4075–4082.

22. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (2019).

23. Duchi, J. C. *Derivations for Linear Algebra and Optimization* in (2016).

24. Dziugaite, G. K., Hsu, K., Gharbieh, W., Arpino, G. & Roy, D. *On the role of data in PAC-Bayes* in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics* (eds Banerjee, A. & Fukumizu, K.) **130** (PMLR, 2021), 604–612. https://proceedings.mlr.press/v130/karolina-dziugaite21a.html.

25. Dziugaite, G. K. & Roy, D. M. *Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data* (2017).

26. Dziugaite, G. K. & Roy, D. M. *Data-dependent PAC-Bayes priors via differential privacy* in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (2018), 8440–8450.

27. Eckart, C. & Young, G. The approximation of one matrix by another of lower rank. *Psychometrika* **1,** 211–218. ISSN: 0033-3123. https://link.springer.com/article/10.1007%2FBF02288367 (1936).

28. Feng, J., Durner, M., Marton, Z.-C., Balint-Benczedi, F. & Triebel, R. *Introspective Robot Perception using Smoothed Predictions from Bayesian Neural Networks* 2021. https://arxiv.org/pdf/2109.12869.

29. Flam-Shepherd, D., Requeima, J. & Duvenaud, D. *Mapping Gaussian process priors to Bayesian neural networks* in *NIPS Bayesian deep learning workshop* **13** (2017).

30. Fortuin, V. *Priors in Bayesian Deep Learning: A Review* 2021. https://arxiv.org/pdf/2105.06868.

31. Gal, Y. & Ghahramani, Z. Bayesian convolutional neural networks with Bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158* (2015).

32. Gawlikowski, J. *et al. A Survey of Uncertainty in Deep Neural Networks* 2021. `https://arxiv.org/pdf/2107.03342`.

33. Germain, P., Bach, F., Lacoste, A. & Lacoste-Julien, S. PAC-Bayesian Theory Meets Bayesian Inference. *Advances in Neural Information Processing Systems* **29.** `https://papers.nips.cc/paper/6569-improving-variational-autoencoders-with-inverse-autoregressive-flow` (2016).

34. Ghosh, S., Yao, J. & Doshi-Velez Finale. Structured Variational Learning of Bayesian Neural Networks with Horseshoe Priors. *International Conference on Machine Learning,* 1744–1753. ISSN: 1938-7228. `http://proceedings.mlr.press/v80/ghosh18a.html` (2018).

35. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).

36. Graves, A. *Practical Variational Inference for Neural Networks* in *Advances in Neural Information Processing Systems* (eds Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F. & Weinberger K. Q.) **24** (Curran Associates, Inc, 2011). `https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf`.

37. Grigorescu, S., Trasnea, B., Cocias, T. & Macesanu, G. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics* **37,** 362–386. ISSN: 1556-4959 (2020).

38. Grosse, R. & Martens, J. A Kronecker-factored approximate Fisher matrix for convolution layers. *International Conference on Machine Learning,* 573–582. ISSN: 1938-7228. `http://proceedings.mlr.press/v48/grosse16.html` (2016).

39. Guedj, B. *A Primer on PAC-Bayesian Learning* in *Proceedings of the French Mathematical Society* (Société Mathématique de France, Lille, France, 2019), 391–414. `https://discovery.ucl.ac.uk/id/eprint/10083910/`.

40. Gulshan, V. *et al.* Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA* **316,** 2402. ISSN: 0098-7484 (2016).

41. Gupta, A. K. & Nagar, D. K. *Matrix variate distributions* ISBN: 9781584880462 (Chapman & Hall/CRC, Boca Raton, Fla., 2000).

42. He, K., Zhang, X., Ren, S. & Sun, J. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification* in *2015 IEEE International Conference on Computer Vision (ICCV)* (IEEE, 2015), 1026–1034. ISBN: 978-1-4673-8391-2.

43. He, K., Zhang, X., Ren, S. & Sun, J. *Deep Residual Learning for Image Recognition* in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2016), 770–778. ISBN: 978-1-4673-8851-1.

44. Hernández-Lobato, J. M. & Adams, R. P. *Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks* in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (JMLR.org, 2015), 1861–1869.

45. Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

46. Hinton, G. E. & van Camp, D. *Keeping the neural networks simple by minimizing the description length of the weights* in *Proceedings of the sixth annual conference on Computational learning theory - COLT '93* (ACM Press, New York, New York, USA, 1993).

47. Hsu, Y.-C., Liu, Y.-C., Ramasamy, A. & Kira, Z. *Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines* (2019).

48. Huang, Y. & Chen, Y. *Survey of State-of-Art Autonomous Driving Technologies with Deep Learning* in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (IEEE, 2020), 221–228. ISBN: 978-1-7281-8915-4.

49. Humt, M., Lee, J. & Triebel, R. in *IROS 2020 Long-Term Autonomy Workshop* ().

50. Ioffe, S. & Szegedy, C. *Batch normalization: Accelerating deep network training by reducing internal covariate shift* in *International conference on machine learning* (2015), 448–456.

51. Jospin, L. V., Buntine, W., Boussaid, F., Laga, H. & Bennamoun, M. Hands-on Bayesian Neural Networks–a Tutorial for Deep Learning Users. *ACM Comput. Surv* **1** (2020).

52. Kao, T.-C., Jensen, K. T., Bernacchia, A. & Hennequin, G. Natural continual learning: success is a journey, not (just) a destination. *arXiv preprint arXiv:2106.08085* (2021).

53. Karaletsos, T. & Bui, T. D. *Hierarchical Gaussian Process Priors for Bayesian Neural Network Weights* (2020).

54. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* in *3rd International Conference on Learning Representations, ICLR 2015* (eds Bengio, Y. & Lecun, Y.) (2015).

55. Kirkpatrick, J. *et al.* Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* **114,** 3521–3526. ISSN: 1091-6490. `https://www.pnas.org/content/114/13/3521.short` (2017).

56. Krishnan, R., Subedar, M. & Tickoo, O. *Efficient Priors for Scalable Variational Inference in Bayesian Deep Neural Networks* in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)* (IEEE, 2019), 773–777. ISBN: 978-1-7281-5023-9.

57. Krishnan, R., Subedar, M. & Tickoo, O. Specifying Weight Priors in Bayesian Deep Neural Networks with Empirical Bayes. *Proceedings of the AAAI Conference on Artificial Intelligence* **34,** 4477–4484. ISSN: 2159-5399 (2020).

58. Kristiadi, A., Hein, M. & Hennig, P. *Being bayesian, even just a bit, fixes overconfidence in relu networks* in *International Conference on Machine Learning* (2020), 5436–5446.

59. Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. *Master's thesis, University of Tront.* `https://ci.nii.ac.jp/naid/20001706980/` (2009).

60. Lai, K., Bo, L., Ren, X. & Fox, D. *A large-scale hierarchical multi-view rgb-d object dataset* in *2011 IEEE international conference on robotics and automation* (2011), 1817–1824.

61. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86,** 2278–2324. ISSN: 00189219 (1998).

62. LeCun, Y. *et al.* Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* **1,** 541–551. ISSN: 0899-7667 (1989).

63. Lee, J., Feng, J., Humt, M., Müller, M. G. & Triebel, R. *Trust Your Robots! Predictive Uncertainty Estimation of Neural Networks with Sparse Gaussian Processes* in *5th Annual Conference on Robot Learning* (2021).

64. Lee, J., Humt, M., Feng, J. & Triebel, R. Estimating Model Uncertainty of Neural Networks in Sparse Information Form. *International Conference on Machine Learning,* 5702–5713. ISSN: 1938-7228. http://proceedings.mlr.press/v119/lee20b.html (2020).

65. Lopez-Paz, D. & Ranzato, M. Gradient episodic memory for continual learning. *Advances in neural information processing systems* **30,** 6467–6476 (2017).

66. Louizos, C. & Welling Max. Structured and Efficient Variational Deep Learning with Matrix Gaussian Posteriors. *International Conference on Machine Learning,* 1708–1716. ISSN: 1938-7228. http://proceedings.mlr.press/v48/louizos16.html (2016).

67. MacKay, D. J. C. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation* **4,** 448–472. ISSN: 0899-7667 (1992).

68. MacKay, D. J. & Mac Kay, D. J. *Information theory, inference and learning algorithms* (Cambridge university press, 2003).

69. Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P. & Wilson, A. G. A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems* **32,** 13153–13164 (2019).

70. Martens, J. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193* (2014).

71. Martens, J. & Grosse, R. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. *International Conference on Machine Learning,* 2408–2417. ISSN: 1938-7228. http://proceedings.mlr.press/v37/martens15.html (2015).

72. McAllester, D. in *Learning Theory and Kernel Machines* 203–215 (Springer, Berlin, Heidelberg, 2003). https://link.springer.com/chapter/10.1007/978-3-540-45167-9_16.

73. McAllester, D. A. *PAC-Bayesian model averaging* in *Proceedings of the twelfth annual conference on Computational learning theory - COLT '99* (ACM Press, New York, New York, USA, 1999).

74. McAllester, D. A. Some PAC-Bayesian Theorems. *Machine Learning* **37,** 355–363. ISSN: 1573-0565. https://link.springer.com/article/10.1023/a:1007618624809 (1999).

75. McAllester, D. A. PAC-Bayesian Stochastic Model Selection. *Machine Learning* **51,** 5–21. ISSN: 1573-0565. https://link.springer.com/article/10.1023/a:1021840411064 (2003).

76. Molchanov, P., Mallya, A., Tyree, S., Frosio, I. & Kautz, J. *Importance estimation for neural network pruning* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), 11264–11272.

77. Nair, V. & Hinton, G. E. *Rectified linear units improve restricted boltzmann machines* in *Proceedings of the 27th International Conference on International Conference on Machine Learning* (2010), 807–814.

78. Nalisnick, E., Hernández-Lobato, J. M. & Smyth, P. *Dropout as a structured shrinkage prior* in *International Conference on Machine Learning* (2019), 4712–4722.

79. Neal, R. M. *MCMC using Hamiltonian dynamics* 2011. https://arxiv.org/pdf/1206.1901.

80. Netzer, Y. *et al.* Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* **2011.** http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf (2011).

81. Opper, M. in *On-Line Learning in Neural Networks* 363–378 (Cambridge University Press, 1999).

82. Papamarkou, T., Hinkle, J., Young, M. T. & Womble, D. Challenges in Markov chain Monte Carlo for Bayesian neural networks. *arXiv e-prints,* arXiv:1910.06539 (2019).

83. Parrado-Hernández, E., Ambroladze, A., Shawe-Taylor, J. & Sun, S. PAC-Bayes bounds with data dependent priors. *The Journal of Machine Learning Research* **13,** 3507–3531 (2012).

84. Paszke, A. *et al.* in *Advances in Neural Information Processing Systems 32* (eds Wallach, H. *et al.*) 8024–8035 (Curran Associates, Inc, 2019). `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

85. Pham, H., Dai, Z., Xie, Q. & Le, Q. V. *Meta Pseudo Labels* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), 11557–11568.

86. Pitas, K. *On {PAC}-Bayes Bounds for Deep Neural Networks using the Loss Curvature* 2020. `https://openreview.net/forum?id=SklgfkSFPH`.

87. Pourzanjani, A. A., Jiang, R. M. & Petzold, L. R. *Improving the identifiability of neural networks for Bayesian inference* in *NIPS Workshop on Bayesian Deep Learning* **4** (2017), 29. `http://bayesiandeeplearning.org/2017/papers/15.pdf`.

88. Quarteroni, A., Sacco, R. & Saleri, F. *Numerical Mathematics* ISBN: 978-1-4757-7394-1 (Springer, 2007).

89. Rebuffi, S.-A., Kolesnikov, A., Sperl, G. & Lampert, C. H. *icarl: Incremental classifier and representation learning* in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (2017), 2001–2010.

90. Richter, S. R., Vineet, V., Roth, S. & Koltun, V. *Playing for Data: Ground Truth from Computer Games* in *European Conference on Computer Vision (ECCV)* (eds Leibe, B., Matas, J., Sebe, N. & Welling, M.) **9906** (Springer International Publishing, 2016), 102–118.

91. Rish, I. in *International Conference on Machine Learning, ICML 2021* (). `https://icml.cc/media/icml-2021/Slides/10833_njzbXvu.pdf`.

92. Ritter, H., Botev, A. & Barber, D. *A Scalable Laplace Approximation for Neural Networks* in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (International Conference on Representation Learning, Vancouver, Canada, 2018). `https://discovery.ucl.ac.uk/id/eprint/10080902/`.

93. Ritter, H., Botev, A. & Barber, D. *Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting* in *Advances in Neural Information Processing Systems* (eds S. Bengio *et al.*) **31** (Curran Associates, Inc, 2018). `https://proceedings.neurips.cc/paper/2018/file/f31b20466ae89669f9741e047487eb37-Paper.pdf`.

94. Rivasplata, O., Parrado-Hernández, E., Shawe-Taylor, J., Sun, S. & Szepesvári, C. *PAC-Bayes bounds for stable algorithms with instance-dependent priors* in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (2018), 9234–9244.

95. Robbins, H. in *Contribution to the Theory of Statistics* 157–164 (University of California Press, 1956). ISBN: 9780520313880.

96. Rusu, A. A. *et al. Progressive Neural Networks* (2016).

97. Senior, A. W. *et al.* Improved protein structure prediction using potentials from deep learning. *Nature* **577,** 706–710. ISSN: 0028-0836 (2020).

98. Shin, H., Lee, J. K., Kim, J. & Kim, J. *Continual learning with deep generative replay* in *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017), 2994–3003.

99. Shinde, K., Lee, J., Humt, M., Sezgin, A. & Triebel, R. *Learning Multiplicative Interactions with Bayesian Neural Networks for Visual-Inertial Odometry* in *Workshop on AI for Autonomous Driving (AIAD), the 37th International Conference on Machine Learning (ICML)* (2020).

100. Sun, P. *et al. Scalability in Perception for Autonomous Driving: Waymo Open Dataset* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020).

101. Sun, S., Chen, C. & Carin, L. Learning Structured Weight Uncertainty in Bayesian Neural Networks. *Artificial Intelligence and Statistics,* 1283–1292. ISSN: 2640-3498. `http://proceedings.mlr.press/v54/sun17b.html` (2017).

102. Sun, S., Zhang, G., Shi, J. & Grosse, R. *Functional Variational Bayesian Neural Networks* 2019. `https://arxiv.org/pdf/1903.05779`.

103. Szegedy, C., Ioffe, S., Vanhoucke, V. & Alemi, A. A. *Inception-v4, inception-resnet and the impact of residual connections on learning* in *Thirty-first AAAI conference on artificial intelligence* (2017).

104. Tajbakhsh, N. *et al.* Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning? *IEEE Transactions on Medical Imaging* **35,** 1299–1312. ISSN: 0278-0062 (2016).

105. Tan, C. *et al. A Survey on Deep Transfer Learning* in *Artificial neural networks and machine learning – ICANN 2018* (eds Kůrková, V., Manolopoulos, Y., Hammer, B., Iliadis, L. & Maglogiannis, I.) **11141** (Springer, Cham, 2018), 270–279. ISBN: 978-3-030-01423-0.

106. Tang, Z. *et al. SKFAC: Training Neural Networks With Faster Kronecker-Factored Approximate Curvature* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), 13479–13487.

107. Thiemann, N., Igel, C., Wintenberger, O. & Seldin, Y. *A Strongly Quasiconvex PAC-Bayesian Bound* in *Proceedings of the 28th International Conference on Algorithmic Learning Theory* (eds Hanneke, S. & Reyzin, L.) **76** (PMLR, 2017), 466–492. `https://proceedings.mlr.press/v76/thiemann17a.html`.

108. Van de Ven, G. M. & Tolias, A. S. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734* (2019).

109. van Erven, T. Pac-bayes mini-tutorial: A continuous union bound. *arXiv preprint arXiv: 1405.1580* (2014).

110. Voulodimos, A., Doulamis, N., Doulamis, A. & Protopapadakis, E. Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience* **2018,** 7068349. ISSN: 1687-5265 (2018).

111. Wang, T. S., Marton, Z.-C., Brucker, M. & Triebel, R. *How Robots Learn to Classify New Objects Trained from Small Data Sets* in *Conference on Robot Learning* (2017), 408–417.

112. Wenzel, F. *et al. How Good is the Bayes Posterior in Deep Neural Networks Really?* in *Proceedings of the 37th International Conference on Machine Learning* (eds III, H. D. & Singh, A.) **119** (PMLR, 2020), 10248–10259. https://proceedings.mlr.press/v119/wenzel20a.html.

113. Wilson, A. G. *The Case for Bayesian Deep Learning* 2020. http://arxiv.org/pdf/2001.10995v1.

114. Wilson, A. G. & Izmailov, P. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. *CoRR* **abs/2002.08791** (2020).

115. Wolf, M. *Mathematical Foundations of Supervised Learning* 2018. https://www-m5.ma.tum.de/foswiki/pub/M5/Allgemeines/MA4801_2018S/ML_notes_main.pdf.

116. Wu, A. *et al.* Deterministic variational inference for robust Bayesian neural networks. *7th International Conference on Learning Representations, ICLR 2019*. http://publications.eng.cam.ac.uk/1195631/ (2019).

117. Xiao, H., Rasul, K. & Vollgraf, R. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms* 2017.

118. Yang, B., Bender, G., Le, Q. V. & Ngiam, J. *CondConv: conditionally parameterized convolutions for efficient inference* in *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (2019), 1307–1318.

119. Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems* **27,** 3320–3328 (2014).

120. Zeng, W. & Urtasun, R. *MLPrune: Multi-Layer Pruning for Automated Neural Network Compression* 2019. https://openreview.net/forum?id=r1g5b2RcKm.

121. Zhou, W., Veitch, V., Austern, M., Adams, R. P. & Orbanz, P. *Non-vacuous Generalization Bounds at the ImageNet Scale: a PAC-Bayesian Compression Approach* in *International Conference on Learning Representations* (2018).

A. Appendix

A.1. Kronecker-Factored Optimal Curvature

The FIM is usually approximated with the empirical distribution over the training data $\mathcal{D}$ and it is computed over batches $B_p = ((\mathbf{x}_k^p, \mathbf{y}_k^p))_{k=1}^{K_p}$ with $(\mathbf{x}_k^p, \_) \sim P$ and $\mathbf{y}_k^p \sim p(\cdot|\mathbf{x}, \boldsymbol{\theta})$ for $p \in [\bar{P}]$. In the following, we use the superscript k to denote the correspondence of the (pre-)activation and their gradients to the sample $(\mathbf{x}_k^p, \mathbf{y}_k^p) \in B_p$ for $p \in [\bar{P}]$. Moreover, we assume the FIM to be block-diagonal. Now, we present K-FOC, which is the proposed method for computing the FIM of deep neural networks. In this section, we first formulate the FIM computations as an optimization problem over the sums of Kronecker products, followed by its connection to the best rank-one problem and the power iteration method as its solution (Section A.1.1). Then, we discuss the derived algorithm (Section A.1.2)

A.1.1. Approximation of Sums of Kronecker Products

The FIM approximated on a batch B_p is a sum of Kronecker-factored matrices for both fully-connected and convolutional layers, namely $\sum_{k=1}^{K_p} \mathcal{D}\mathbf{s}^k (\mathcal{D}\mathbf{s}^k)^T \otimes \frac{1}{K_p} \bar{\mathbf{a}}^k (\bar{\mathbf{a}}^k)^T$ and $\sum_{k=1}^{K_p} \sum_{\mathbf{t} \in \mathcal{T}} \sum_{\mathbf{t}' \in \mathcal{T}} \mathcal{D}\mathbf{s}^k{}_{\mathbf{t}} (\mathcal{D}\mathbf{s}^k)^T_{\mathbf{t}'} \otimes \frac{1}{K_p} \bar{\mathbf{a}}^k_{:,\mathbf{t}} (\bar{\mathbf{a}}^k_{:,\mathbf{t}'})^T$. The resulting approximation of the FIM is in general not Kronecker-factored. Nonetheless, a Kronecker-factored approximation brings many advantages like low memory consumption, easy computations of the inverse, and a fast sampling from the corresponding matrix variate distribution [71]. Hence, we aim to find Kronecker-factors that better approximate the FIM. This boils down to approximating a sum of Kronecker-factored matrices (similar to Kao *et al.* [52]) by one Kronecker-factored matrix, *i.e.* for $M, N, K \in \mathbb{N}$, $\mathbf{L}^k \in \mathbb{R}^{M \times M}$ and $\mathbf{R}^k \in \mathbb{R}^{N \times N}$ for $k \in [K]$ we aim to find:

$$\hat{\mathbf{L}}, \hat{\mathbf{R}} \in \underset{\mathbf{L} \in \mathbb{R}^{M \times M}, \mathbf{R} \in \mathbb{R}^{N \times N}}{\arg\min} \| \sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L} \otimes \mathbf{R} \|_F. \tag{A.1}$$

In general, a solution of Equation A.1 is not unique. This can be seen by scaling the left factor $\hat{\mathbf{L}}$ by $\alpha \neq 0$ and $\hat{\mathbf{R}}$ by $\frac{1}{\alpha}$. Thus, in the following, we additionally assume that $\hat{\mathbf{L}}$ is normalized, $\|\hat{\mathbf{L}}\|_F = 1$. The choice that $\hat{\mathbf{L}}$ is normalized is arbitrary and one could achieve

all results presented in this work when instead $\hat{\mathbf{R}}$ would be normalized. As it turns out, the problem is equivalent to a rank-one approximation as proven in the lemma below.

Lemma A.1.1. *Let $M, N, K \in \mathbb{N}$, $\mathbf{L}^k \in \mathbb{R}^{M\times M}$ and $\mathbf{R}^k \in \mathbb{R}^{N\times N}$ for $k \in [K]$. Then*

$$\|\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\|_F = \|\sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L})\operatorname{vec}(\mathbf{R})^T\|_F. \tag{A.2}$$

Proof. Let $i, j \in [MN]$. Then the i,j-th entry of the left matrix is

$$\begin{aligned}\left(\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\right)_{i,j} &= \sum_{k=1}^{K} \mathbf{L}^k_{i_1,j_1}\mathbf{R}^k_{i_2,j_2} - \mathbf{L}_{i_1,j_1}\mathbf{R}_{i_2,j_2}\\ &= \left(\sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L})\operatorname{vec}(\mathbf{R})^T\right)_{M(i_1-1)+j_1,N(i_2-1)+j_2},\end{aligned} \tag{A.3}$$

with $i = N(i_1-1)+i_2$, $j = N(j_1-1)+j_2$. Hence, both matrices have the same entries and only the order of the entries is in general different. Therefore, the sum over the squared entries and thus the Frobenius norm is the same

$$\begin{aligned}\|\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\|_F^2 &= \sum_{i=1}^{M^2}\sum_{j=1}^{N^2}\left(\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\right)^2_{i,j}\\
&= \sum_{i_1,j_1=1}^{N}\sum_{i_2,j_2=1}^{M}\left(\sum_{k=1}^{K} \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L}\otimes\mathbf{R}\right)^2_{N(i_1-1)+i_2,N(j_1-1)+j_2}\\
&= \sum_{i_1,j_1=1}^{N}\sum_{i_2,j_2=1}^{M}\left(\sum_{k=1}^{K} \mathbf{L}^k_{i_1,j_1} \otimes \mathbf{R}^k_{i_2,j_2} - \mathbf{L}_{i_1,j_1}\otimes\mathbf{R}_{i_2,j_2}\right)^2\\
&= \sum_{i_1,j_1=1}^{N}\sum_{i_2,j_2=1}^{M}\left(\sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L})\operatorname{vec}(\mathbf{R})^T\right)^2_{M(i_1-1)+j_1,N(i_2-1)+j_2}\\
&= \|\sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T - \operatorname{vec}(\mathbf{L})\operatorname{vec}(\mathbf{R})^T\|_F^2\end{aligned} \tag{A.4}$$

□

Therefore, one can use the power method to solve this problem. Nonetheless, each step of the plain power method consists of a matrix multiplication with an $M^2 \times N^2$-sized matrix, which is in general not feasible for layers of modern neural networks. The complexity can be reduced by utilizing that the matrix is a sum of a few rank-one matrices. This is shown in Algorithm 2. With this, the convergence properties of the power method are achieved

with a computational complexity of $\mathcal{O}\left(n^{max}K(N^2+M^2)\right)$. Also, only $\mathcal{O}\left(K(N^2+M^2)\right)$ memory is needed. Here, we assume that a matrix multiplication $\mathbf{AB}$ for $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times k}$ has the complexity $\mathcal{O}(mnk)$ while a Hadamard product $\mathbf{A} \odot \mathbf{C}$ for $\mathbf{C} \in \mathbb{R}^{m\times n}$ can be computed in $\mathcal{O}(mn)$. The correctness of Algorithm 2 is shown in Lemma A.1.2.

Lemma A.1.2. *Let* $\mathbf{A} = \sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k)\operatorname{vec}(\mathbf{R}^k)^T$ *and* $\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$ *be its singular value decomposition with* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ *and* $\mathbf{u}_i^T\mathbf{u}_j = \mathbf{v}_i^T\mathbf{v}_j = \mathbb{1}[i=j]$. *Then there is a solution of Equation A.1 with*

$$\operatorname{vec}(\hat{\mathbf{L}}) = \mathbf{u}_1, \operatorname{vec}(\hat{\mathbf{R}}) = \sigma_1\mathbf{v}_1. \tag{A.5}$$

If $\sigma_1 > \sigma_2$, *the solution is unique up to changing the sign of both factors and Algorithm 2 converges almost surely to this solution.*

Proof. The main idea of the proof is to use Lemma A.1.1 to identify the problem with a best rank-one approximation. The algorithm then corresponds to the power method that utilizes the Kronecker factorization for a faster and memory-efficient computation of the matrix-vector products in the Kronecker matrix space.

By the Eckart–Young–Mirsky theorem [27], an optimal rank-one approximation for A in the Frobenius norm is

$$\sigma_1\mathbf{u}_1\mathbf{v}_1^T \in \underset{\hat{\mathbf{A}}\in\mathbb{R}^{M^2\times N^2}:\operatorname{rank}(\hat{\mathbf{A}})=1}{\arg\min} \|\mathbf{A}-\hat{\mathbf{A}}\|_F, \tag{A.6}$$

which is unique up to changing the sign of both factors if $\sigma_1 > \sigma_2$.

Therefore, the matrices $\hat{\mathbf{L}}$ and $\hat{\mathbf{R}}$ that satisfy Equation A.5 are optimal solutions of Equation A.1. Moreover, the left factor is normalized, *e.g.* $\|\hat{\mathbf{L}}\|_F = \|\mathbf{u}_1\|_2 = 1$.

The equivalence of Algorithm 2 with the power method can be seen by multiplying $\mathbf{AA}^T$ with $\operatorname{vec}(\mathbf{L}^{(n-1)})$ for $\mathbf{L}^{(n-1)} \in \mathbb{R}^{N^2}$:

$$\begin{aligned}
\mathbf{A}^T\operatorname{vec}(\mathbf{L}^{(n-1)}) &= \sum_{k=1}^{K} \operatorname{vec}(\mathbf{R}^k)\operatorname{vec}(\mathbf{L}^k)^T\operatorname{vec}(\mathbf{L}^{(n-1)}) && \text{(A.7)}\\
&= \sum_{k=1}^{K} \langle \mathbf{L}^k, \mathbf{L}^{(n-1)}\rangle_F \operatorname{vec}(\mathbf{R}^k) && \text{(A.8)}\\
&= \operatorname{vec}(\bar{\mathbf{R}}^{(n)}) && \text{(A.9)}
\end{aligned}$$

and

$$\mathbf{A}\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n-1)}) = \sum_{k=1}^{K} \operatorname{vec}(\mathbf{L}^k) \operatorname{vec}(\mathbf{R}^k)^T \operatorname{vec}(\bar{\mathbf{R}}^{(n)}) \tag{A.10}$$

$$= \sum_{k=1}^{K} \langle \mathbf{R}^k, \bar{\mathbf{R}}^{(n)} \rangle_F \operatorname{vec}(\mathbf{L}^k) \tag{A.11}$$

$$= \|\bar{\mathbf{R}}^{(n)}\|_F \sum_{k=1}^{K} \langle \mathbf{R}^k, \mathbf{R}^{(n)} \rangle_F \operatorname{vec}(\mathbf{L}^k) \tag{A.12}$$

$$= \|\bar{\mathbf{R}}^{(n)}\|_F \operatorname{vec}(\bar{\mathbf{L}}^{(n)}). \tag{A.13}$$

Hence, Algorithm 2 computes the same iterations as the standard power method:

$$\frac{\mathbf{A}\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n-1)})}{\|\mathbf{A}\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n-1)})\|_2} = \frac{\|\bar{\mathbf{R}}^{(n)}\|_F \operatorname{vec}(\bar{\mathbf{L}}^{(n)})}{\|\bar{\mathbf{R}}^{(n)}\|_F \|\bar{\mathbf{L}}^{(n)}\|_F} = \frac{\operatorname{vec}(\bar{\mathbf{L}}^{(n)})}{\|\bar{\mathbf{L}}^{(n)}\|_F} = \operatorname{vec}(\mathbf{L}^{(n)}). \tag{A.14}$$

The final right factor then corresponds to $\mathbf{A}^T \operatorname{vec}(\mathbf{L}^{(n)}) \approx \sigma_1 \mathbf{v}_1$. For $\sigma_1 > \sigma_2$, the convergence properties are inherited from the power method, see *e.g.* [5]. □

Remark. Lemma A.1.2 shows that Algorithm 2 is equivalent to the power method for the corresponding best rank-one problem. Therefore, convergence rates and error bounds are directly inherited from the power method, see *e.g.* [88]. Even in the case when the first singular value is not (much) larger than the other singular values and no convergence is achieved, the resulting matrices of Algorithm 2 are with high probability in the span of the singular vectors corresponding to the set of large singular values [9]. Hence, in this case, the approximation will still converge to good Kronecker factors with high probability.

A.1.2. Kronecker-Factored Optimal Curvature

Now, we present K-FOC for both fully-connected and convolutional layers. For fully-connected layers, one can apply Algorithm 2 with $\mathbf{L}^k = \mathcal{D}\mathbf{s}^k(\mathcal{D}\mathbf{s}^k)^T$ and $\mathbf{R}^k = \bar{\mathbf{a}}^k(\bar{\mathbf{a}}^k)^T$ for $k \in [K_p]$ for batch $p \in [\bar{P}]$ to obtain the optimal Kronecker factors for this batch. The resulting computational complexity is $\mathcal{O}\left(n^{max} K_p (d_{l-1}^2 + d_l^2)\right)$ with a memory consumption of $\mathcal{O}\left(K_p(d_{l-1}^2 + d_l^2)\right)$. Compared to fully-connected layers, convolutional layers involve a sum over $|\mathcal{T}|^2 = h_l^2 w_l^2$ Kronecker factors in Equation 2.3. Hence, the complexity of directly applying Algorithm 2 with $\mathbf{L}^{k,\mathbf{t},\mathbf{t}'} = \mathcal{D}\mathbf{s}^k{}_{\mathbf{t}}(\mathcal{D}\mathbf{s}^k)^T_{\mathbf{t}'}$ and $\mathbf{R}^{k,\mathbf{t},\mathbf{t}'} = \bar{\mathbf{a}}^k_{:,\mathbf{t}}(\bar{\mathbf{a}}^k_{:,\mathbf{t}'})^T$ would be $\mathcal{O}\left(n^{max} K_p |\mathcal{T}|^2 ((c_{l-1}|\Delta| + 1)^2 + c_l^2)\right)$ and would use $\mathcal{O}\left(K_p |\mathcal{T}|^2 ((c_{l-1}|\Delta| + 1)^2 + c_l^2)\right)$ memory, which is usually not feasible especially for high-resolution images. To reduce the complexity, one can incorporate that each Kronecker factor is an outer product of two vectors and that the summation is over all combinations of $\mathbf{t} \in \mathcal{T}$ and $\mathbf{t}' \in \mathcal{T}$. With this, Line 7 of Algorithm 2 can be computed as

Algorithm 2: Power method for sums of Kronecker products

input :
- $(\mathbf{L}^k)_{k\in[K]}$ left matrices
- $(\mathbf{R}^k)_{k\in[K]}$ right matrices
- $n^{max} = 100$ maximal number of steps
- $\delta = 10^{-5}$ stopping precision

output: $\hat{\mathbf{L}}, \hat{\mathbf{R}} \in \arg\min_{\mathbf{L},\mathbf{R}} \| \sum_{k=1}^K \mathbf{L}^k \otimes \mathbf{R}^k - \mathbf{L} \otimes \mathbf{R}\|_F$

1 **function** `kronecker_power_method`$((\mathbf{L}^k)_{k\in[K]}, (\mathbf{R}^k)_{k\in[K]})$
2 $\quad \mathrm{vec}(\bar{\mathbf{L}}^{(0)}) \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ `standard normal initialization of` $\bar{\mathbf{L}}^{(0)}$
3 $\quad \mathbf{L}^{(0)} \leftarrow \frac{\bar{\mathbf{L}}^{(0)}}{\|\bar{\mathbf{L}}^{(0)}\|_F}$ ▷ `normalize` $\bar{\mathbf{L}}^{(0)}$
4 $\quad$ **for** $n \in [n^{max}]$ **do**
5 $\quad\quad \bar{\mathbf{R}}^{(n)} \leftarrow \sum_{k=1}^K \langle \mathbf{L}^k, \mathbf{L}^{(n-1)} \rangle_F \mathbf{R}^k$ ▷ `first power iteration step`
6 $\quad\quad \mathbf{R}^{(n)} \leftarrow \frac{\bar{\mathbf{R}}^{(n)}}{\|\bar{\mathbf{R}}^{(n)}\|_F}$ ▷ `normalize` $\bar{\mathbf{R}}^{(n)}$
7 $\quad\quad \bar{\mathbf{L}}^{(n)} \leftarrow \sum_{k=1}^K \langle \mathbf{R}^k, \mathbf{R}^{(n)} \rangle_F \mathbf{L}^k$ ▷ `second power iteration step`
8 $\quad\quad \mathbf{L}^{(n)} \leftarrow \frac{\bar{\mathbf{L}}^{(n)}}{\|\bar{\mathbf{L}}^{(n)}\|_F}$ ▷ `normalize` $\bar{\mathbf{L}}^{(n)}$
9 $\quad\quad$ **if** $\|\mathbf{L}^{(n)} - \mathbf{L}^{(n-1)}\|_F < \delta$ **then**
10 $\quad\quad\quad$ **break**
11 $\quad \hat{\mathbf{L}} \leftarrow \mathbf{L}^{(n)}$
12 $\quad \hat{\mathbf{R}} \leftarrow \sum_{k=1}^K \langle \mathbf{L}^k, \mathbf{L}^{(n)} \rangle_F \mathbf{R}^k$ ▷ `first power iteration step`

$$\sum_{k=1}^{K_p} \sum_{\mathbf{t},\mathbf{t}'\in\mathcal{T}} \langle \mathcal{D}\mathbf{s}^k{}_{\mathbf{t}} (\mathcal{D}\mathbf{s}^k)^T_{\mathbf{t}'}, \mathbf{L}^{(n-1)} \rangle_F \bar{\mathbf{a}}^k_{:,\mathbf{t}} (\bar{\mathbf{a}}^k_{:,\mathbf{t}'})^T = \sum_{k=1}^{K_p} (\mathcal{D}\mathbf{s}^k)^T (\bar{\mathbf{a}}^k)^T \mathbf{L}^{(n-1)} \bar{\mathbf{a}}^k \mathcal{D}\mathbf{s}^k. \tag{A.15}$$

Line 5 can be computed similarly. Altogether, one can pre-compute the matrix

$$\mathbf{X}^k \leftarrow (\bar{\mathbf{a}}^k \mathcal{D}\mathbf{s}^k)^T \text{ for } k \in [K] \tag{A.16}$$

in advance and in each iteration, the lines 5 and 7 can be replaced by

$$\bar{\mathbf{R}}^{(n)} \leftarrow \sum_{k=1}^{K_p} \mathbf{X}^k \mathbf{L}^{(n-1)} (\mathbf{X}^k)^T \quad \text{and} \quad \bar{\mathbf{L}}^{(n)} \leftarrow \sum_{k=1}^{K_p} (\mathbf{X}^k)^T \mathbf{R}^{(n)} \mathbf{X}^k, \tag{A.17}$$

respectively. The full adaption of Algorithm 2 for convolutions is shown in Algorithm 3.

The computational complexity hence reduces to $\mathcal{O}(K_p(c_{l-1}|\Delta|+1)|\mathcal{T}|c_l + n^{max} K_p(c_{l-1}|\Delta|+1)\, c_l((c_{l-1}|\Delta|+1)+c_l))$. Moreover, only $\mathcal{O}\left((c_{l-1}|\Delta|+1)^2 + c_l^2 + K_p|\mathcal{T}|((c_{l-1}|\Delta|+1)+c_l)\right)$

Algorithm 3: Power method for convolutions

input :
- $(\mathcal{D}\mathbf{s}^k)_{k\in[K]}$ pre-activation derivatives
- $(\bar{\mathbf{a}}^k)_{k\in[K]}$ activations
- $n^{max} = 100$ maximal number of steps
- $\delta = 10^{-5}$ stopping precision

output: $\hat{\mathbf{L}}, \hat{\mathbf{R}} \in \arg\min_{\mathbf{L},\mathbf{R}} \|\frac{1}{k}\sum_{k=1}^{K}\sum_{\mathbf{t}\in\mathcal{T}}\sum_{\mathbf{t}'\in\mathcal{T}} \mathcal{D}\mathbf{s}^k{}_{\mathbf{t}}(\mathcal{D}\mathbf{s}^k)^T_{\mathbf{t}'} \otimes \bar{\mathbf{a}}^k_{:,\mathbf{t}}(\bar{\mathbf{a}}^k_{:,\mathbf{t}'})^T - \mathbf{L}\otimes\mathbf{R}\|_F$

function `kronecker_power_method_convolutions`$((\mathcal{D}\mathbf{s}^k)_{k\in[K]}, \bar{\mathbf{a}}^k)_{k\in[K]})$

- $\mathrm{vec}(\bar{\mathbf{L}}^{(0)}) \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ `standard normal initialization of` $\bar{\mathbf{L}}^{(0)}$
- $\mathbf{L}^{(0)} \leftarrow \frac{\bar{\mathbf{L}}^{(0)}}{\|\bar{\mathbf{L}}^{(0)}\|_F}$ ▷ `normalize` $\bar{\mathbf{L}}^{(0)}$
- $\mathbf{X}^k \leftarrow (\mathcal{D}\mathbf{s}^k)^T\bar{\mathbf{a}}^k$ for $k \in [K]$ ▷ `pre-compute` $\mathbf{X}^k$
- **for** $n \in [n^{max}]$ **do**
  - $\bar{\mathbf{R}}^{(n)} \leftarrow \sum_{k=1}^{K} \mathbf{X}^k\mathbf{L}^{(n-1)}(\mathbf{X}^k)^T$ ▷ `first power iteration step`
  - $\mathbf{R}^{(n)} \leftarrow \frac{\bar{\mathbf{R}}^{(n)}}{\|\bar{\mathbf{R}}^{(n)}\|_F}$ ▷ `normalize` $\bar{\mathbf{R}}^{(n)}$
  - $\bar{\mathbf{L}}^{(n)} \leftarrow \sum_{k=1}^{K}(\mathbf{X}^k)^T\mathbf{R}^{(n)}\mathbf{X}^k$ ▷ `second power iteration step`
  - $\mathbf{L}^{(n)} \leftarrow \frac{\bar{\mathbf{L}}^{(n)}}{\|\bar{\mathbf{L}}^{(n)}\|_F}$ ▷ `normalize` $\bar{\mathbf{L}}^{(n)}$
  - **if** $\|\mathbf{L}^{(n)} - \mathbf{L}^{(n-1)}\|_F < \delta$ **then**
    - **break**
- $\hat{\mathbf{L}} \leftarrow \mathbf{L}^{(n)}$
- $\hat{\mathbf{R}} \leftarrow \frac{1}{k}\sum_{k=1}^{K_p} \mathbf{X}^k\mathbf{L}^{(n)}(\mathbf{X}^k)^T$ ▷ `first power iteration step`

memory is needed. This is of the same order as K-FAC if $|\mathcal{T}| > n^{max}\max\{c_{l-1}|\Delta|+1, c_l\}$. Furthermore, we note that the adaption of the power method for convolutions can also be applied to fully-connected layers by viewing them as convolutional layers with $\mathcal{T} = \{(1,1)\}$. While the complexity usually increases to $\mathcal{O}\left(K_p d_{l-1}d_l + n^{max}K_p d_{l-1}d_l(d_{l-1}+d_l)\right)$ with this approach, the memory needed is reduced for large batch sizes to $\mathcal{O}\left(d_{l-1}^2 + d_l^2 + K_p(d_{l-1}+d_l)\right)$.

So far, with this method, optimal factors for a batch of samples can be found with practical complexity and memory consumption. However, the optimal factors $(\mathbf{L}^p, \mathbf{R}^p)$ for the batches $B_p, p \in [\bar{P}]$ additionally need to be aggregated to obtain the FIM. Similar to K-FAC, independence between different batches can be assumed and the FIM could be estimated as $\mathbf{F}_l \approx (\sum_{p=1}^{P}\mathbf{L}^p)\otimes(\frac{1}{\bar{P}}\sum_{p=1}^{\bar{P}}\mathbf{R}^p)$, denoted as K-FOC_approx in the following. Empirically, this does not hold in general and the approximation quality worsens for small batch sizes. Another possibility is to collect all factors in one step. In a second step the optimal factors $\hat{\mathbf{L}}, \hat{\mathbf{R}} \in \arg\min_{\mathbf{L},\mathbf{R}} \|\sum_{p=1}^{\bar{P}}\mathbf{L}^p\otimes\frac{1}{\bar{P}}\mathbf{R}^p - \mathbf{L}\otimes\mathbf{R}\|_F$ can be computed using the power method and the estimate of the FIM is given by $\mathbf{F}_l \approx \hat{\mathbf{L}}\otimes\hat{\mathbf{R}}$. A drawback of this method is that the factors of all batches need to be kept in memory.

A trade-off between approximation quality and memory consumption is approximating the running average $\hat{\mathbf{L}}, \hat{\mathbf{R}} \in \arg\min_{\mathbf{L},\mathbf{R}} \|\hat{\mathbf{L}}^{p-1}\otimes\frac{p-1}{p}\hat{\mathbf{R}}^{p-1} + \mathbf{L}^p\otimes\frac{1}{p}\mathbf{R}^p - \mathbf{L}\otimes\mathbf{R}\|_F$ with $\hat{\mathbf{L}}^1 = \mathbf{L}^1, \hat{\mathbf{R}}^1 = \mathbf{R}^1$ for each batch. The FIM can then be approximated as $\mathbf{F}_l \approx \hat{\mathbf{L}}^{\bar{P}}\otimes\hat{\mathbf{R}}^{\bar{P}}$ and

only the running average and the current matrices need to be kept in memory resulting in the same memory consumption as K-FOC_approx with a small additional computational overhead. We denote this variant of K-FOC as K-FOC_running. Next, we evaluate the proposed approaches empirically.

A.1.3. Experiments and Results

We now present our experiments, which are designed to check the approximation quality of the K-FOC when compared to the K-FAC. To compute the FIM, we draw one sample from the model's predictive distribution for each data point and use the same samples for the ground truth block-diagonal FIM as for all approximations. To this end, we compare the relative Frobenius error with respect to the ground-truth FIM. All experiments are implemented in PyTorch [84] on top of the curvature library [49, 64, 99] and are run on an Nvidia RTX 3060 GPU.

Fully-Connected Layers

Experiment Setup. To validate the method on fully-connected layers, we train a fully-connected neural network with one hidden layer on the three UCI datasets Boston Housing, Concrete Compression Strength and Energy Efficiency, following Hernández-Lobato & Adams [44] and Lee *et al.* [64]. Similar to their work, the hidden layer also consists of 50 units. For all three datasets, the FIM is computed over the full dataset.

Results. The first row of Figure A.1 shows the results of the fully-connected network for the Boston Housing dataset. The results on the other UCI datasets show similar qualitative behavior and are shown in Appendix A.1.3. For each method, the mean error is depicted with a thick line in addition to the minimal and maximal value among the ten runs in the light area. One can observe that K-FAC is strictly better in terms of the Frobenius error compared to the diagonal approximation. The K-FOC approximation with the independence assumptions (K-FOC_approx) usually improves with increasing batch size and surpasses K-FAC starting at around a batch size of 10. The K-FOC approximation that estimates the running average (K-FOC_running) performs best even though the difference between the two K-FOC methods diminishes for large batch sizes. The runtime for this experiment is analyzed in Appendix A.1.3.

Convolutional Layers

Experiment Setup. For convolutional layers, the method is validated on the first two convolutional layers of a LeNet-5 [62] architecture on the MNIST dataset [61]. The FIM and its approximations are computed over the training split of the dataset.

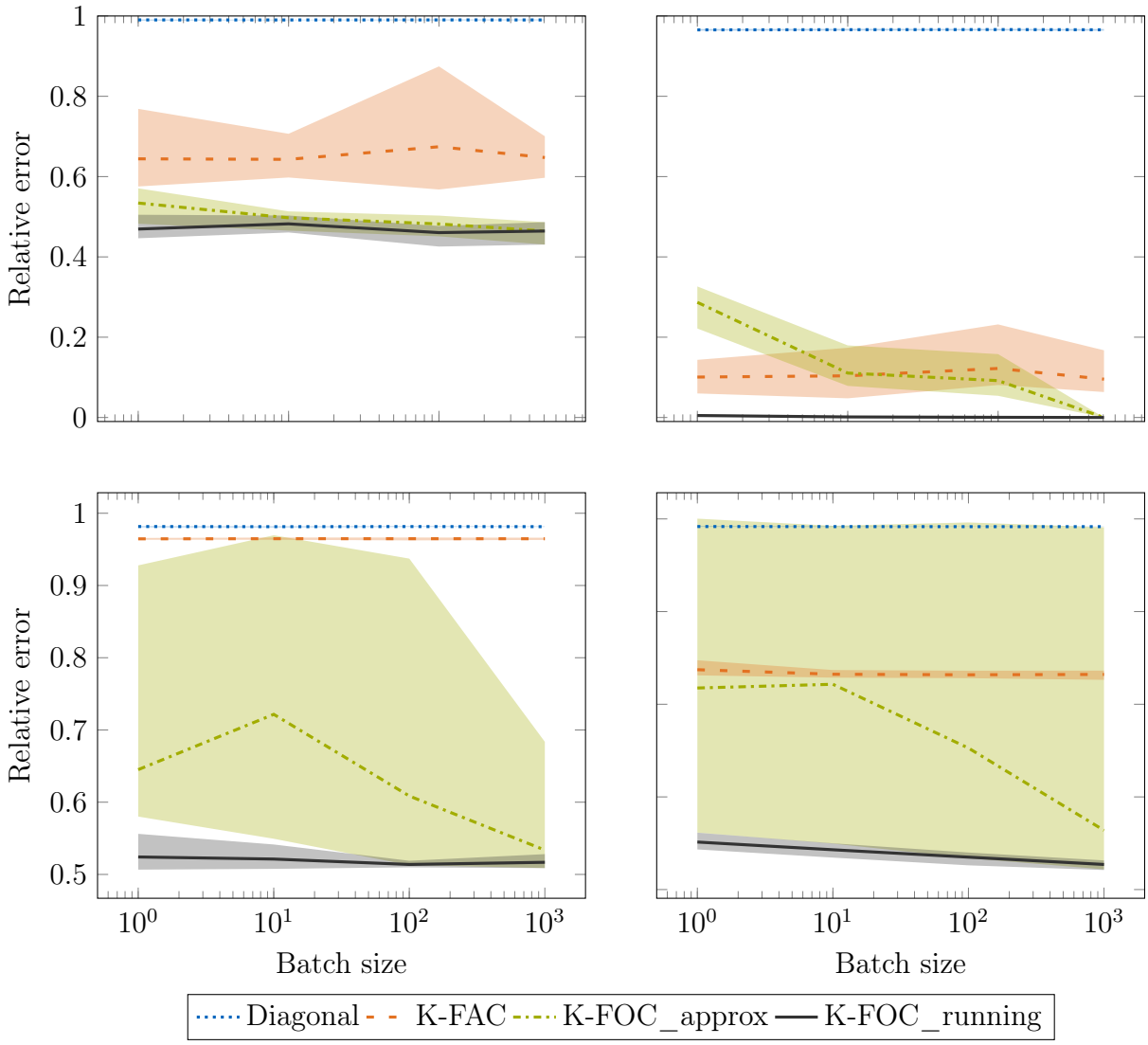


Figure A.1.: Comparison of the relative Frobenius error for diagonal approximations (blue), K-FAC (orange), and K-FOC (green and gray) for different batch sizes for the first (left) and second (right) layer of the relative network architecture. The two fully-connected layers trained on the Boston Housing dataset are in the first row and the convolutional layers learned on MNIST are in the second row. The light area shows the minimal and maximal values among ten independent runs while the thick lines are the mean errors. For K-FOC, both aggregation strategies as described in Section A.1.2 are depicted

Table A.1.: Runtime in ms

		Batch size			
Name	Approximation	1	10	100	full
Boston Housing	K-FAC	180.7±5.5	18.5±1.7	3.2±0.8	1.3±1.1
	K-FOC_approx	727.1±12.8	118.0±7.5	22.9±10.8	3.0±0.8
	K-FOC_running	1262.6±14.5	175.6±6.8	25.2±7.4	2.6±1.0
Concrete Compression Strength	K-FAC	366.8±19.0	35.8±2.9	4.0±1.1	0.2±2.1
	K-FOC_approx	1462.3±17.0	225.2±17.8	37.2±10.8	1.4±1.3
	K-FOC_running	2503.6±34.6	314.6±11.9	55.0±17.6	1.5±2.7
Energy Efficiency	K-FAC	266.7±20.4	27.2±3.4	3.6±1.6	0.7±1.5
	K-FOC_approx	1072.7±15.8	256.7±40.1	28.4±8.1	4.5±5.4
	K-FOC_running	1908.1±20.4	346.5±29.0	41.1±11.9	4.5±4.7

Results. In the second row of Figure A.1, one can see the relative error on the first two convolutional layers. Similar to the fully-connected experiment, one can observe that K-FOC_approx is usually closer to the block-diagonal matrix than the K-FAC and the diagonal approximation. Nonetheless, the approximation quality highly depends on the specific run and the variance between the runs is in general high. K-FOC_running consistently outperforms all other methods by a large margin and has a lower variance than K-FOC_approx. For both K-FOC methods, the relative error decreases with an increasing batch size, which is not the case for K-FAC or the diagonal method.

Runtime

Table A.1 shows the runtime of each approximation method on the UCI datasets. K-FOC_approx and K-FOC_running need around seven and nine times as much computational time compared to K-FAC, respectively. In general, each iteration of the power iteration has a similar runtime as K-FAC but usually, multiple iterations are needed to converge which then corresponds to a multiple of the runtime of K-FAC. In return, this shows that usually much fewer iterations are needed than the maximal number of steps $n^{max} = 100$. The additional runtime from K-FOC_running compared to K-FOC_approx comes from the aggregation of the factors for different batches utilizing again the power method to compute an estimate of the running mean. Still, both K-FOC algorithms are in the same complexity classes as K-FAC and only have a small linear overhead compared to K-FAC.

Further Results for K-FOC

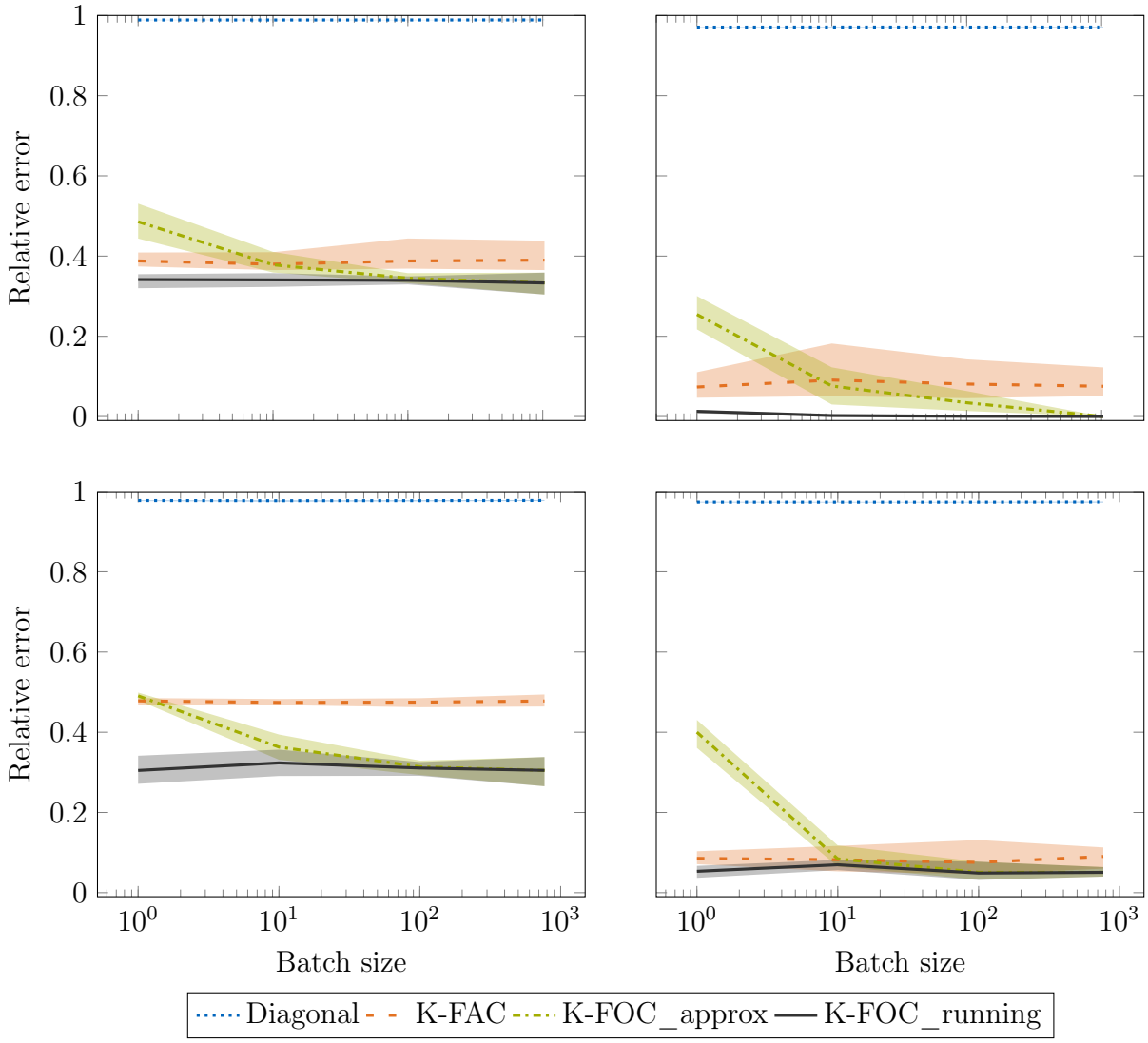


Figure A.2.: Comparison of the relative Frobenius error for diagonal approximations (blue), K-FAC (orange), and K-FOC (green and gray) for different batch sizes for the first (left) and second (right) layer of the relative network architecture. The rows correspond to the fully-connected layers trained on the Concrete Compression Strength (top) and Energy Efficiency dataset (bottom), respectively. The light area shows the minimal and maximal values among ten independent runs while the thick lines are the mean errors. For K-FOC, both aggregation strategies as described in Section A.1.2 are depicted.

A.2. Further Algorithms

This section contains the pseudo-code to compute the objective function (Algorithm 4), the curvature scaling (Algorithm 5), and the PAC-Bayesian bounds (Algorithm 6 and Algorithm 7). Additionally, the pseudo-code for progressive neural networks with an arbitrary network architecture is given in Algorithm 8.

Algorithm 4: Parameter objective.

input :

$\mathcal{I}$	index set
$f.$	network architecture
$(\boldsymbol{\theta}_\mathbf{i})_{\mathbf{i}\in\mathcal{I}}$	parameters
$\mathcal{D}$	dataset
$(\hat{\boldsymbol{\theta}}_\mathbf{i})_{\mathbf{i}\in\mathcal{I}}$	prior mean
$(\tilde{\mathbf{F}}_\mathbf{i})_{\mathbf{i}\in\mathcal{I}}$	prior precision
τ	temperature scaling
ε	PAC-Bayes probability

output : objective value v

function `objective`$(\mathcal{I}, f., (\boldsymbol{\theta}_\mathbf{i})_{\mathbf{i}\in\mathcal{I}}, \mathcal{D}, (\hat{\boldsymbol{\theta}}_\mathbf{i})_{\mathbf{i}\in\mathcal{I}}, (\tilde{\mathbf{F}}_\mathbf{i})_{\mathbf{i}\in\mathcal{I}}, \tau, \varepsilon)$

- $N \leftarrow$ `len`$(\mathcal{D})$
- $nll \leftarrow -\ln(\mathcal{D}|f_{\boldsymbol{\theta}})$
- $q \leftarrow \frac{1}{2\tau}\sum_{\mathbf{i}\in\mathcal{I}}(\boldsymbol{\theta}_\mathbf{i} - \tilde{\boldsymbol{\theta}}_\mathbf{i})^T\tilde{\mathbf{F}}_\mathbf{i}(\boldsymbol{\theta}_\mathbf{i} - \tilde{\boldsymbol{\theta}}_\mathbf{i})$
- $v \leftarrow \begin{cases} nll + q & \text{MAP objective} \\ \texttt{pac\_bayes\_bound}(\frac{nll}{\ln 2N}, q, N, \varepsilon) & \text{PAC-Bayes objective} \end{cases}$

Algorithm 5: Approximate upper bound for the curvature scaling.

input :

$\mathcal{I}$	index set
$((\alpha_\mathbf{i}, \beta_\mathbf{i}))_{\mathbf{i}\in\mathcal{I}}$	curvature scales
$(\Lambda_\mathbf{i})_{\mathbf{i}\in\mathcal{I}}$	set of eigenvalue sets
nll	negative log-likelihood
q	quadratic penalty term
N	dataset length
τ	temperature scaling
$n^{(\leq t-1)}$	number of fixed weights
ε	PAC-Bayes probability

output : curvature scaling value v

function `curvature_scaling`$(\mathcal{I}, ((\alpha_\mathbf{i}, \beta_\mathbf{i}))_{\mathbf{i}\in\mathcal{I}}, (\Lambda_\mathbf{i})_{\mathbf{i}\in\mathcal{I}}, nll, q, N, \tau, n^{(\leq t-1)}, \varepsilon)$

- $eel \leftarrow \frac{1}{N\ln 2}\left(-nll + \frac{\tau}{2}\left(n^{(\leq t-1)} + \sum_{\mathbf{i}\in\mathcal{I}}\sum_{\lambda\in\Lambda_\mathbf{i}}\frac{\tau}{\beta_\mathbf{i}+\alpha_\mathbf{i}\lambda^{-1}}\right)\right)$
- $kl \leftarrow q + \frac{1}{2}\left(\sum_{\mathbf{i}\in\mathcal{I}}\sum_{\lambda\in\Lambda_\mathbf{i}}\frac{1}{\beta_\mathbf{i}\lambda+\alpha_\mathbf{i}} - 1 - \ln(\beta_\mathbf{i}\lambda + \alpha_\mathbf{i})\right)$
- $v \leftarrow$ `pac_bayes_bound`$(eel, kl, N, \varepsilon)$

Algorithm 6: PAC-Bayes McAllester Bound.

input :
- e expected empirical error
- kl KL-divergence
- N dataset length
- ε PAC-Bayes probability

output : value of the bound v

function `pac_bayes_bound_mcallester`(e, kl, N, ε)

$v \leftarrow e + \sqrt{\frac{kl + \ln(\frac{2N}{\varepsilon})}{2N}}$

Algorithm 7: PAC-Bayes Catoni Bound.

input :
- e expected empirical error
- kl KL-divergence
- N dataset length
- ε PAC-Bayes probability

output : value of the bound v

function `pac_bayes_bound_catoni`(e, kl, N, ε)

$v \leftarrow \min_{c>0} \frac{1-\exp(-ce-\frac{kl-\ln\varepsilon}{N})}{1-\exp(-c)}$

Algorithm 8: Progressive neural network.

input :
- $f.$ network architecture
- $\mathfrak{L}$ set of lateral connections

output : callable PNN object that maps the feature $\mathbf{x} \in \mathcal{X}$ to the output of column j, $\mathbf{a}_L^{(j)}$

class `PNN`$(f., \mathfrak{L})$

▷ `Computes the output of column` j `with parameters` $\boldsymbol{\theta}^{(\leq t)}$

function $\texttt{call}^{j}_{\boldsymbol{\theta}^{(\leq t)}}(x)$

for $i \in [j]$ **do**

▷ `Save` $\mathbf{a}_l^{(i)}$ `for` $l \in \mathfrak{L}$

`save_activations`$(f_{\boldsymbol{\theta}^{(i)}}, \mathfrak{L})$

▷ `Compose the layers` $l \in \mathfrak{L}$ `with an aggregation_layer`

`insert_layers`$(f_{\boldsymbol{\theta}^{(i)}}, \mathfrak{L}, \texttt{aggregation\_layer})$

$\mathbf{a}_L^{(i)} \leftarrow f_{\boldsymbol{\theta}^{(i)}}(x)$

return $\mathbf{a}_L^{(j)}$

A.3. Further Results

A.3.1. Small-Scale Continual Learning Experiment

Figure A.3, Figure A.4, and Figure A.5 show the entropy histograms on the small-scale continual learning experiment for PNN, PNN with MC dropout, and PBNN, respectively. Each sample of every dataset is binned by its entropy for the outputs of the different columns. A classifier should have a low uncertainty on ID samples and a high entropy for OOD samples. This is reflected in the entropy histogram by a peak close to zero for the corresponding dataset and a peak far from zero for a different dataset. For all columns but the last, PNN (Figure A.3) produce high peaks on ID data. However, especially the last two columns are also relatively certain on OOD samples which leads to highly overlapping plots between ID and OOD histograms. PBNN (Figure A.5) suffer from the same problem as they also put little uncertainty on OOD samples, especially for the fourth column. Compared to the other two approaches, PNN with MC dropout leads to overall better-separated entropy histograms. Still, the separation gets worse for later columns.

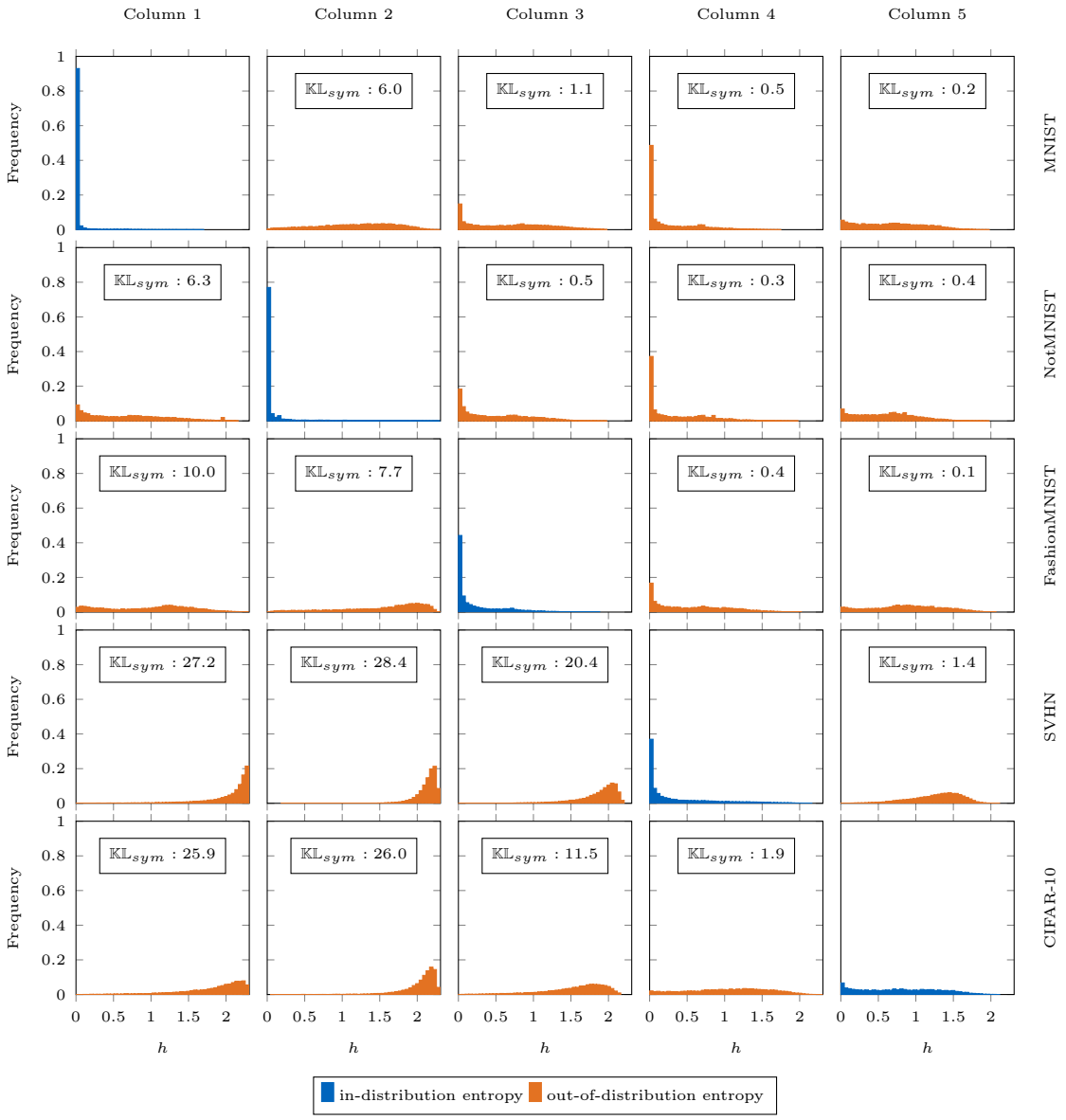


Figure A.3.: The entropy histogram is shown for each column and task for PNN for the small-scale continual learning experiment. A good separation is given if the ID entropy (blue) is smaller than the OOD entropy (orange). The first three columns have a well-separated entropy on the SVHN and CIFAR-10 dataset. On the other tasks, these columns have a slightly worse separation. For the last two columns, a separation between the ID and OOD entropy is not given. In particular, the distribution of column four on the MNIST and NotMNIST dataset has even lower entropy than on its corresponding task.

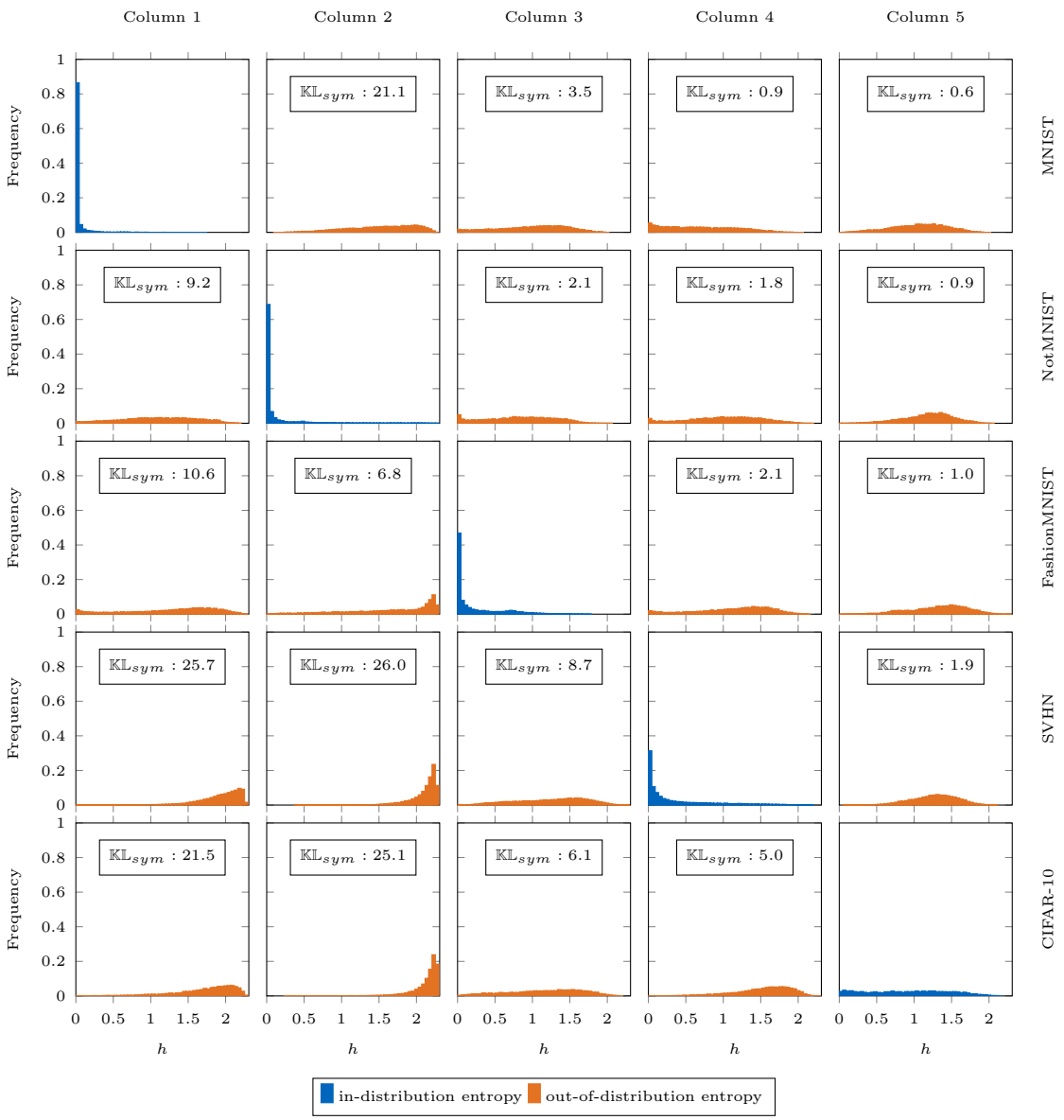


Figure A.4.: The entropy histogram is shown for each column and task for PNN with MC dropout for the small-scale continual learning experiment. A good separation is given if the ID entropy (blue) is smaller than the OOD entropy (orange). For the first three columns, the ID entropy has a high peak close to zero while the out-of distribution entropy is spread and mostly far from zero. While the peak reduces for the last two tasks, the ID entropy is in general still closer to zero than the OOD entropy.

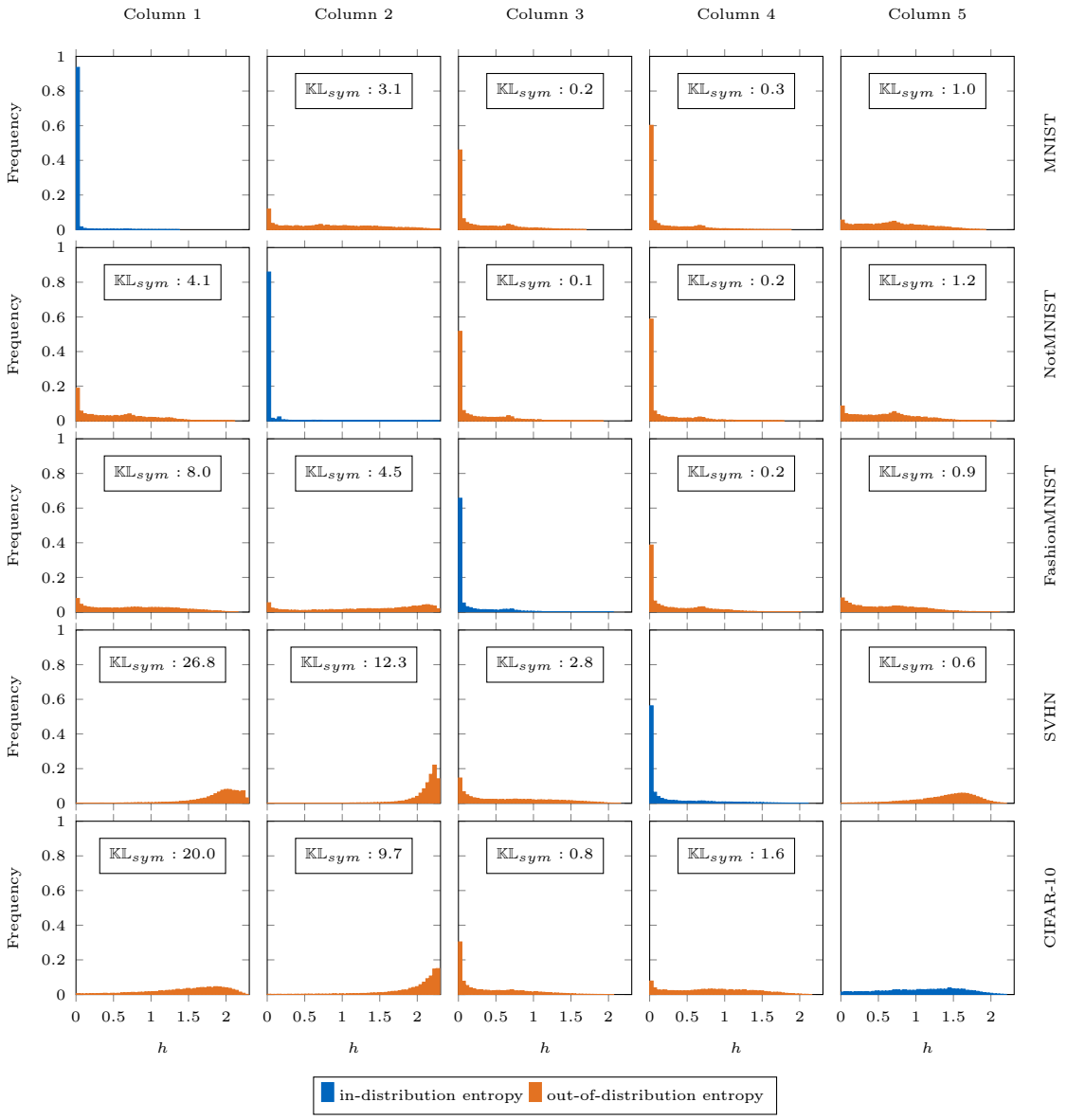


Figure A.5.: The entropy histogram is shown for each column and task for PBNN for the small-scale continual learning experiment. A good separation is given if the ID entropy (blue) is smaller than the OOD entropy (orange). The first two tasks show a good separation between the ID entropy and the OOD entropy on SVHN and CIFAR-10. The entropy on the other datasets intersects with the ID entropy but is on average higher. On the other columns, no separation between the ID and OOD is observable. Especially the last column is more certain on the first three tasks than on the corresponding CIFAR-10 dataset.

A.3.2. Large-Scale Continual Learning Experiment

The entropy histograms for the large-scale continual learning experiment are shown in Figure A.6, Figure A.7, Figure A.8 for PNN, PNN with MC dropout and PBNN, respectively. Here, we use a logarithmic scale on the x-axis to better illustrate the separation, especially for small values. For the first column, PNN and PNN with MC dropout have a better separation between ID and OOD entropy than PBNN. Still, the entropy is already well-split for PBNN. For the other tasks, the separation is enhanced by PBNN compared to the other two methods. In particular, both PNN methods have columns where ID samples have on average more uncertainty than some OOD samples. This does not happen for PBNN.

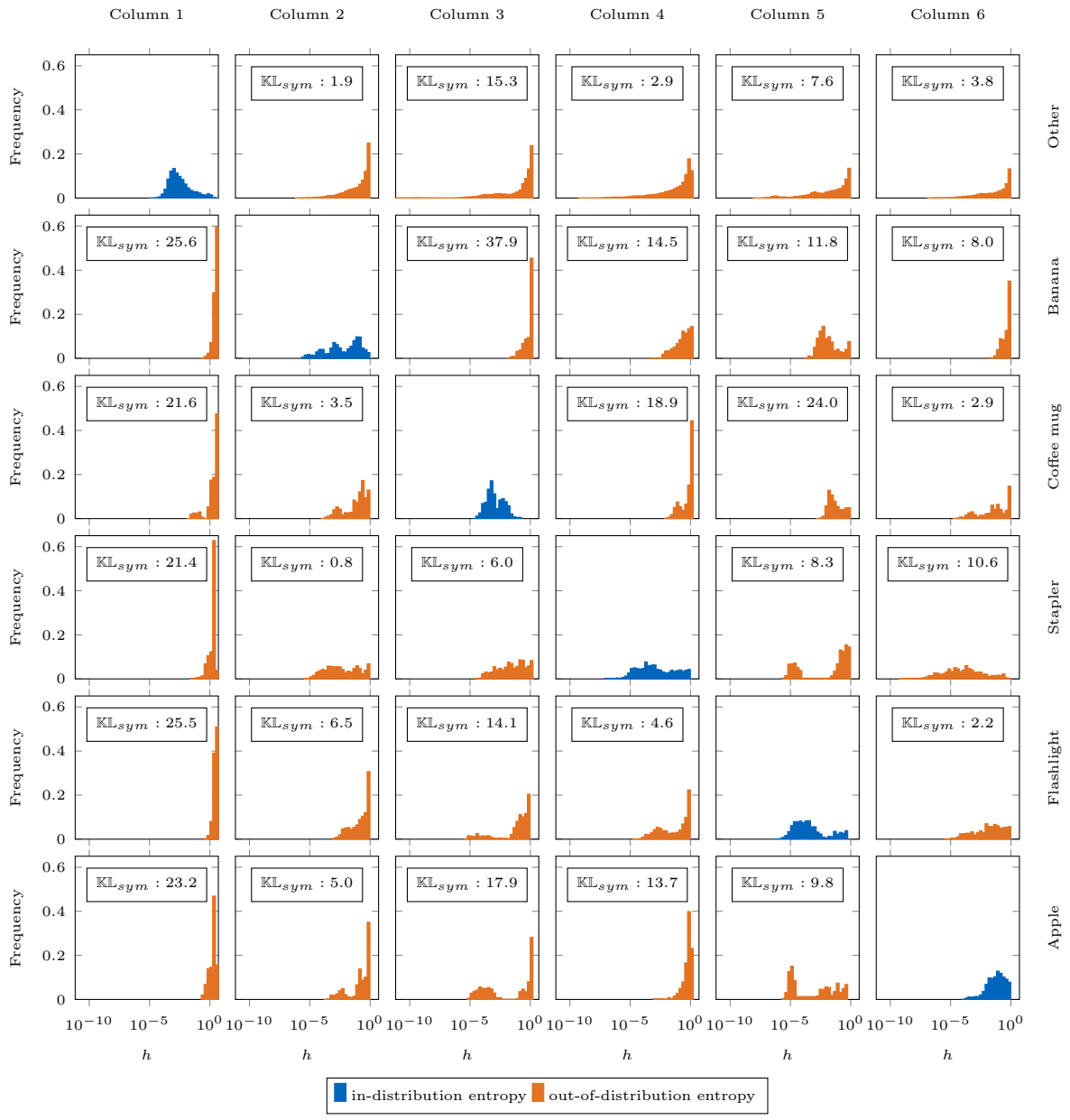


Figure A.6.: The entropy histogram is shown for each column and task for PNN for the large-scale continual learning experiment. A good separation is given if the ID entropy (blue) is smaller than the OOD entropy (orange). While the entropy on the first task is well-separated from the OOD entropy on the other tasks for the first column, the other columns have overlapping regions between the ID and the OOD entropy.

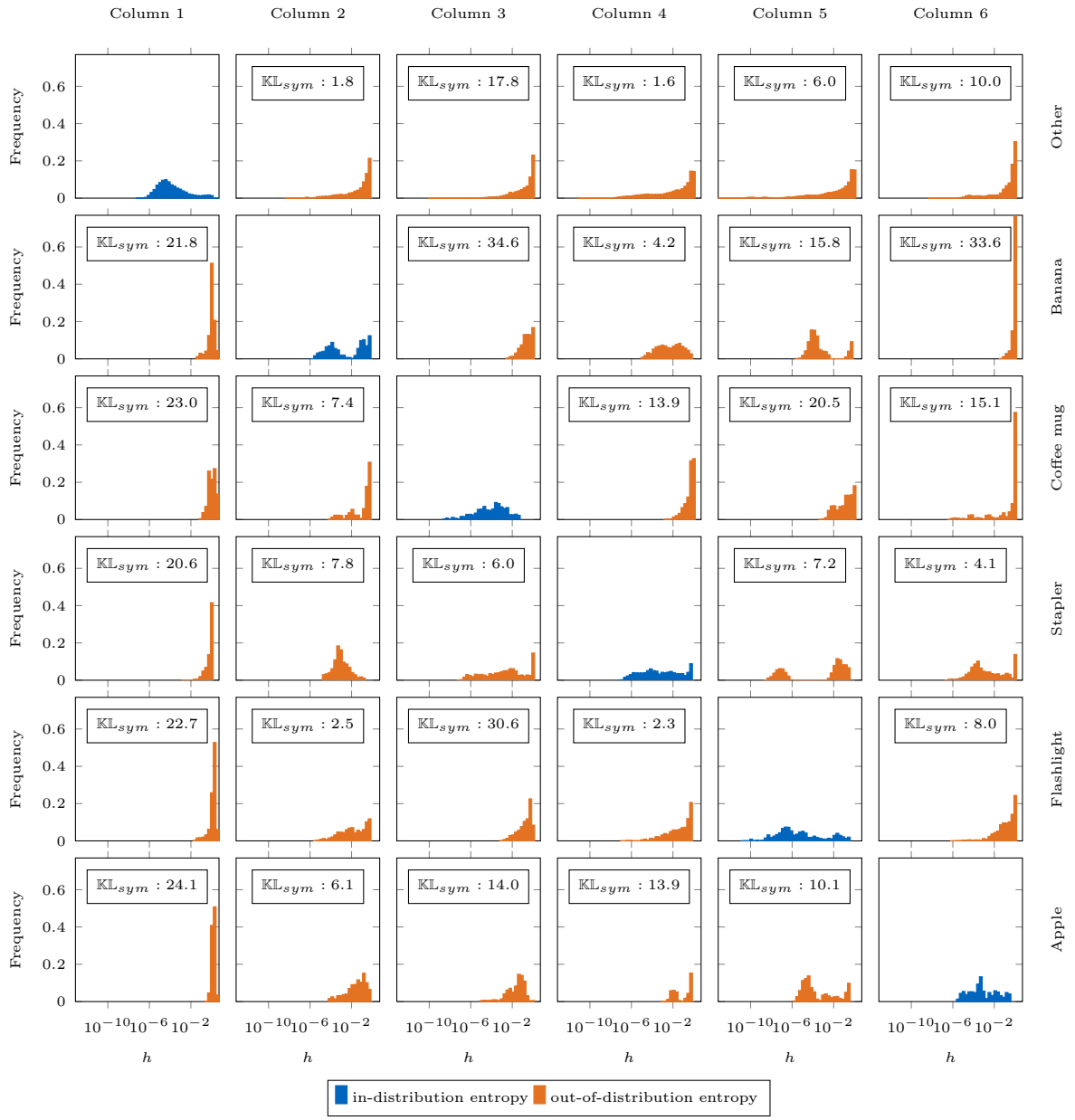


Figure A.7.: The entropy histogram is shown for each column and task for PNN with MC dropout for the large-scale continual learning experiment. A good separation is given if the ID entropy (blue) is smaller than the OOD entropy (orange). The first column puts a high uncertainty on the OOD entropy and hence achieves a good separation. Similarly, the second column has one subset of the banana dataset that is well-separated from the out-distribution samples. Still, the remaining set has a similar entropy as the OOD entropy which leads to a bad separation. In return, it also happens that a subset of a different dataset is given a small entropy like for instance the stapler dataset for column five. In general, this model does not achieve a consistent separation between ID and OOD samples.

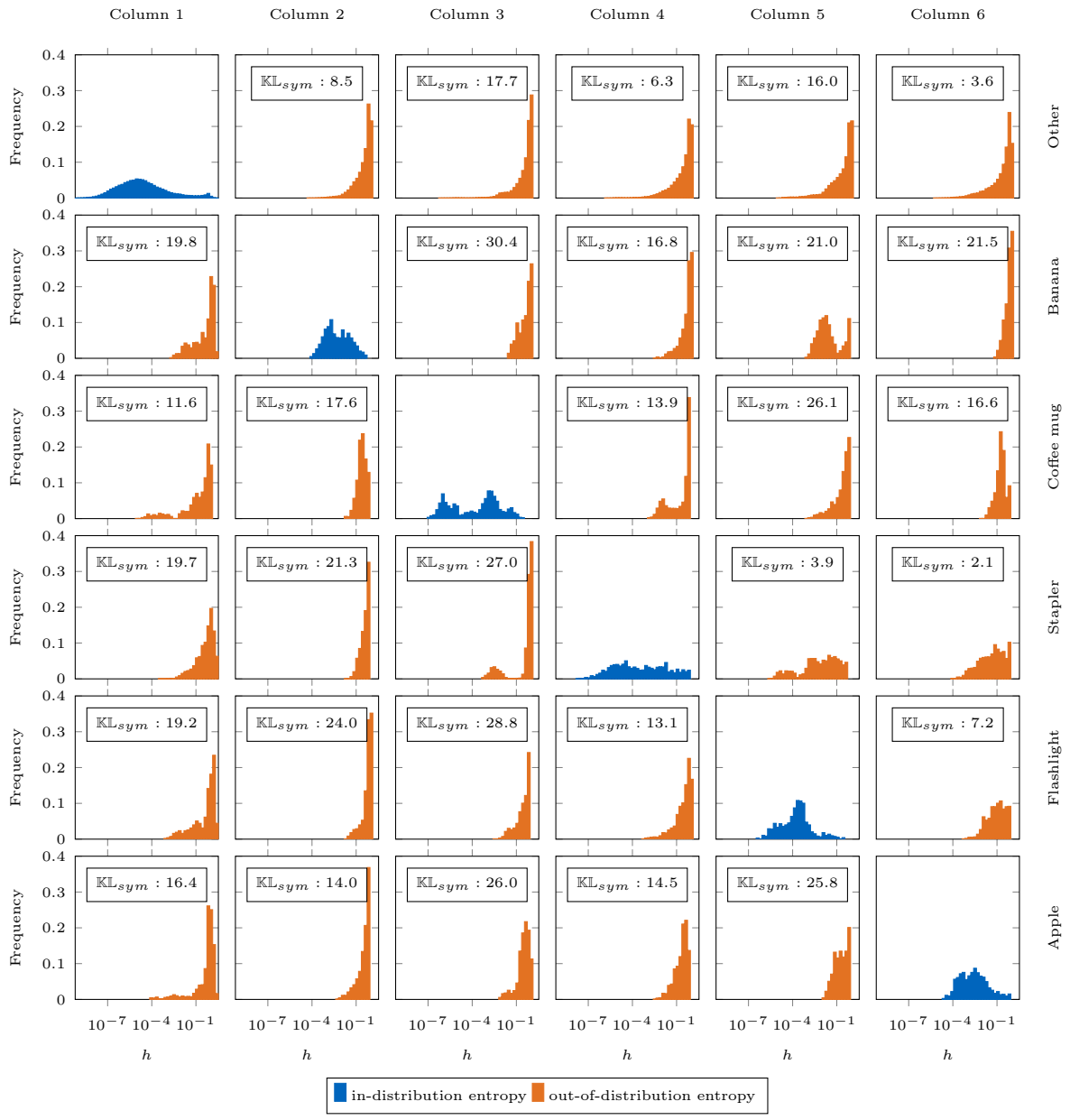


Figure A.8.: The entropy histogram is shown for each column and task for PBNN for the large-scale continual learning experiment. A good separation is given if the ID entropy (blue) is smaller than the OOD entropy (orange). For all columns, the ID entropy of PBNN slightly overlaps with the OOD entropy. However, most samples can be separated well and the peak entropy for ID samples is always smaller than the peak of OOD samples.