Received: 5 May 2022 | Revised: 26 August 2022 | Accepted: 20 September 2022 | Published on: 18 October 2022

DOI: 10.1002/mrm.29485

RESEARCH ARTICLE

Magnetic Resonance in Medicine

Deep, deep learning with BART

Moritz Blumenthal[1] | **Guanxiong Luo**[1] | **Martin Schilling**[1] |
H. Christian M. Holme[2] | **Martin Uecker**[1,2,3,4,5]

[1]Institute for Diagnostic and Interventional Radiology, University Medical Center Göttingen, Göttingen, Germany

[2]Institute of Biomedical Imaging, Graz University of Technology, Graz, Austria

[3]Cluster of Excellence "Multiscale Bioimaging: from Molecular Machines to Networks of Excitable Cells" (MBExC), University of Göttingen, Göttingen, Germany

[4]German Centre for Cardiovascular Research (DZHK), Partner Site Göttingen, Göttingen, Germany

[5]BioTechMed-Graz, Graz, Austria

Correspondence
Martin Uecker, Institute of Biomedical Imaging, Graz University of Technology, Stremayrgasse 16/3, 8010 Graz, Austria.
Email: uecker@tugraz.at

Funding information
Deutsche Forschungsgemeinschaft, Grant/Award Numbers: 432680300 - SFB 1456, EXC2067/1-390729940, UE 189/1-1; Deutsches Zentrum für Herz-Kreislauf-Forschung; National Institutes of Health, Grant/Award Number: U24EB029240; Volkswagen Foundation

Purpose: To develop a deep-learning-based image reconstruction framework for reproducible research in MRI.

Methods: The BART toolbox offers a rich set of implementations of calibration and reconstruction algorithms for parallel imaging and compressed sensing. In this work, BART was extended by a nonlinear operator framework that provides automatic differentiation to allow computation of gradients. Existing MRI-specific operators of BART, such as the nonuniform fast Fourier transform, are directly integrated into this framework and are complemented by common building blocks used in neural networks. To evaluate the use of the framework for advanced deep-learning-based reconstruction, two state-of-the-art unrolled reconstruction networks, namely the Variational Network and MoDL, were implemented.

Results: State-of-the-art deep image-reconstruction networks can be constructed and trained using BART's gradient-based optimization algorithms. The BART implementation achieves a similar performance in terms of training time and reconstruction quality compared to the original implementations based on TensorFlow.

Conclusion: By integrating nonlinear operators and neural networks into BART, we provide a general framework for deep-learning-based reconstruction in MRI.

KEYWORDS
automatic differentiation, deep learning, image reconstruction, inverse problems, MRI, parallel imaging

1 | INTRODUCTION

In the last decades, MRI has advanced substantially in terms of acquisition speed and image quality. Parallel imaging utilizes the signal of multiple receiver coils for image reconstruction by combining the signals in k-space[1-3] or image space.[4] Another step toward the current state-of-the-art image reconstruction was the use of compressed sensing for MRI.[5,6] Advanced methods now integrate compressed sensing and parallel imaging by using sparsifying regularization terms when solving the inverse problem for parallel imaging.[6,7] These tech-

[Correction added after publication 17 November 2022. Due to a publisher's error, the Data Availability Statement has been restored in this version.]

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.
© 2022 The Authors. *Magnetic Resonance in Medicine* published by Wiley Periodicals LLC on behalf of International Society for Magnetic Resonance in Medicine.

niques admit a Bayesian interpretation where regularization terms can be understood as the integration of prior knowledge into the reconstruction.

In recent years, deep learning has become a major research interest in image reconstruction with the goal to improve upon the previously used hand-crafted regularization terms by learning image properties from large datasets. The public availability of deep learning frameworks such as TensorFlow[8] or PyTorch[9] simplifies access to deep learning methods for MRI researchers. Moreover, public datasets from https://mridata.org[10,11] and from the fastMRI challenge[12] provide a large amount of training data and open the field of research to data scientists not having access to MRI data.

Neural networks have been utilized in various ways for MRI reconstruction. Some authors have proposed to learn a direct mapping from the acquired k-space data to the image domain.[13] However, these methods usually lack a data-consistency guarantee, that is, the output of the reconstruction may not be consistent with the measured k-space data. Others have used neural network to train regularizers which can be used for image reconstruction in a subsequent step.[14] In one class of such regularizers, a neural network is trained to enhance an initial reconstruction. Afterwards, the ℓ_2 difference to this reconstruction is used as regularizer.[15,16] Another class of networks with data consistency are networks that model an unrolled iterative optimization algorithm.[17-19] In each iteration of such a network, the network part, usually a CNN or U-Net, updates the current reconstruction and afterwards soft data consistency is imposed by a gradient step or proximal mapping. The resulting unrolled networks are then trained as an end-to-end mapping from the k-space to the image domain.

BART[20] is an open-source framework providing implementations of various calibration methods and reconstruction algorithms for parallel imaging and compressed sensing. It consists of programming libraries and command line tools for easy but flexible access to the programming libraries. BART is developed with the purpose of facilitating reproducible research and has a focus on backwards compatibility, while still offering rapid prototyping and testing of advanced reconstruction algorithm with the goal of translating them into clinical reconstruction pipelines. The high-level reconstruction algorithms of BART are built around programming libraries offering generic implementations of various iterative algorithms as well as an efficient numerical backend. The backend provides functions acting on multidimensional arrays (or tensors) which support acceleration by multiple threads or (multiple) graphical processing units (GPUs). In this work, we extend BART with a complete framework for nonlinear operators. The framework builds on our previous work on nonlinear calibrationless parallel imaging[21] and physics-based reconstruction,[22] and is now extended with automatic differentiation, additional building blocks for neural networks, and new optimization algorithms.[23] In combination with the powerful numerical backend, the nonlinear operator framework can then be used to efficiently train neural networks. Moreover, nonlinear operators can be used to wrap around TensorFlow graphs, allowing the integration of pretrained networks into BART's reconstruction algorithms.[24] MRI reconstruction networks imposing data consistency require a large amount of domain specific knowledge. By integrating neural networks into BART, we benefit from BART's rich set of MRI specific modules and algorithms which can be easily reused for deep learning-based MRI reconstructions. Written in C and only depending on a few external libraries, we consider BART a solid basis for future research that integrates classical image reconstruction with deep learning.

In the remainder of this article we first describe in detail the implementation of our deep learning framework and its integration into BART. There, we focus on the numerical backend, the automatic differentiation, the iterative training algorithms and the neural network framework. Afterwards, we present our implementation of the Variational Network (VarNet)[17] and MoDL,[18] and compare their performance to the original implementations based on TensorFlow.

2 | METHODS

A neural network is a nonlinear function F mapping the input data $\mathbf{x}$ and weights $\boldsymbol{\theta}$ to an output $\mathbf{y} = F(\mathbf{x}; \boldsymbol{\theta})$. Training a neural network corresponds to fitting the neural network to a training dataset by minimizing some suitable loss L, that is,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \left[\sum_i L(\mathbf{y}_i, F(\mathbf{x}_i; \boldsymbol{\theta})) \right]. \tag{1}$$

Usually, neural networks are constructed from small building blocks such as fully connected layers, convolutional layers or activation functions. Automatic differentiation is used to compute the gradients of the loss needed for gradient-based optimization algorithms such as stochastic gradient descent or ADAM.[25] Offering automatic differentiation and efficient implementations for the small building blocks are key features of deep learning frameworks. In the first part of this section, we describe the integration of programming libraries used for deep learning in BART, before we describe our implementations of VarNet and MoDL in the second part.

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

2.1 | Libraries for deep learning in BART

The basic integration of libraries used for neural networks in BART is depicted in Figure 1. The backend provide access to optimized numerical functions. Based on the backend, the nonlinear operator framework is used to construct neural networks from small building blocks and provides automatic differentiation to compute gradients. The nn-library then extends this nonlinear operator framework by deep learning specific functions. Finally, new training algorithms for deep learning are integrated in BART's iterative framework.

2.1.1 | Numerical backend

The numerical backend of BART is designed around functions acting on multidimensional (md) arrays. An md-array is described by its dimensions $\mathbf{d} \in \mathbb{N}^N$, its rank N and, optionally, its strides $\mathbf{s} \in \mathbb{Z}^N$ describing how an element of the md-array is accessed in memory. The offset of an element at position $\mathbf{p} \in \mathbb{N}^N$ is given by $o = \mathbf{p} \cdot \mathbf{s}$. By default, BART assumes column-major ordering, that is, the first dimension of the md-array is stored continuously in memory corresponding to the strides $\mathbf{s}_i = \prod_{j=0}^{i-1} \mathbf{d}_\mathbf{j}$. By manipulating the strides, different views on the memory can be generated without copying data. The memory for an md-array can be allocated on the CPU or on the GPU. On supported GPUs, the GPU memory can be oversubscribed, that is, GPU memory is automatically swapped by the driver to CPU memory.

2.1.2 | Md-functions

Md-functions provide a consistent and flexible interface to functions acting on md-arrays. They loop over all positions defined by the dimensions and apply a scalar-valued kernel on the elements accessed using the provided strides. For example, `md_fmac2` applies

$$\text{for } \mathbf{p} \in \{0, \ldots, \mathbf{d}_0 - 1\} \times \cdots \times \{0, \ldots, \mathbf{d}_{N-1} - 1\} : \\ a[\mathbf{p} \cdot \mathbf{s}^a] \leftarrow a[\mathbf{p} \cdot \mathbf{s}^a] + b[\mathbf{p} \cdot \mathbf{s}^b] \cdot c[\mathbf{p} \cdot \mathbf{s}^c] \,. \quad (2)$$

By setting the strides correspondingly, many functions such as convolutions, matrix-vector multiplications or a dot product can be derived. For example, if $\mathbf{s}^a = 0$, the dot product of b and c is accumulated in $a[0]$. If the memory of an md-array is located on the GPU, the computation of a md-function is automatically executed on the GPU. The loops of the md-functions are generically optimized in the backend. Further, strides corresponding to specific operations such as matrix-matrix multiplication or convolution are detected, and specialized code—possibly from external libraries such as cuBLAS or cuDNN—is executed. Thus, md-functions provide a generic but still efficient interface to the numeric backend.

2.1.3 | Bitwise reproducibility

Floating point arithmetic is not associative making multithreaded programs nondeterministic if the order of the operations depends on the runtime of individual threads. BART's GPU kernels and the generic parallelization do not introduce any nondeterministic operations except for the gridding code of the nuFFT. cuBLAS and cuDNN are deterministic across runs when executed on GPUs with the same architecture except for some specific functions. By default, BART makes only use of these deterministic functions, however, the compile-time option `NON_DETERMINISTIC = 1` can be used to allow BART

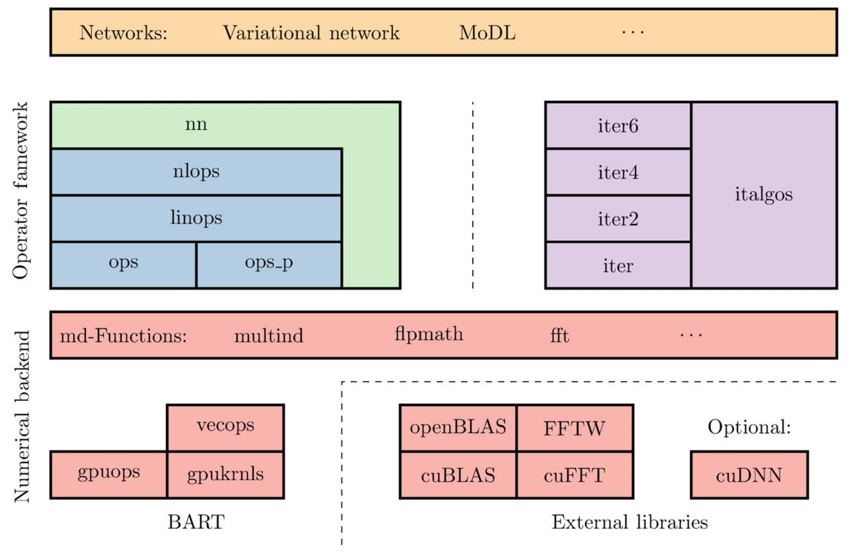


FIGURE 1 Integration of deep learning modules into BART. The numerical backend (red) is accessed by md-functions which invoke BART's internal generically optimized functions or external libraries offering highly optimized code for special functions. Differentiable neural networks are implemented as nonlinear operators (blue). The nn-library (green) extends the nonlinear operator framework by deep learning specific features. The training algorithms are integrated in BART's iterative framework (violet). Iter6 provides a new interface for batched gradient-based training algorithms.

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

to select nondeterministic algorithms to improve computational performance.

2.1.4 | Automatic differentiation and the nonlinear operator framework

Usually, neural networks are trained by gradient-based methods. The vanilla version of a gradient descent algorithm optimizes Eq. (1) by the iteration $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta\nabla_{\boldsymbol{\theta}}\left(\sum_i L(\mathbf{y}_i, F(\mathbf{x}_i;\boldsymbol{\theta}))\right)$, where η is the learning rate and $\nabla_{\boldsymbol{\theta}}$ denotes the gradient of the loss with respect to the weights $\boldsymbol{\theta}$. Automatic differentiation, as described below, allows to construct an operator computing $\sum_i L(\mathbf{y}_i, F(\mathbf{x}_i;\boldsymbol{\theta}))$ and to compute its derivative, that is, gradient, with respect to the weights.

We first describe the automatic differentiation framework on an abstract level for real-valued operators, before we describe the extension to complex variables and the implementation details in subsequent paragraphs. A nonlinear operator (`nlop`) F consists of the forward operator F itself and its (Fréchet)-derivative $\mathrm{D}F|_{\mathbf{x}}$ - a linear operator (`linop`) applying the Jacobian matrix $J|_{\mathbf{x}}$ evaluated at some position $\mathbf{x}$ on its input, that is,

$$F : \mathbb{R}^N \to \mathbb{R}^M \quad \mathrm{D}F|_{\mathbf{x}} : \mathbb{R}^N \to \mathbb{R}^M \quad J_{ij} = \frac{\partial F_i}{\partial x_j}$$
$$\mathbf{x} \mapsto \mathbf{y} = F(\mathbf{x}) \quad \mathrm{d}\mathbf{x} \mapsto \mathrm{d}\mathbf{y} = J|_{\mathbf{x}}\mathrm{d}\mathbf{x}. \tag{3}$$

Usually, the Jacobian J is not stored explicitly, instead, the derivative $\mathrm{D}F|_{\mathbf{x}}$ or its transposed $\mathrm{D}F^T|_{\mathbf{x}}$ can be applied on test inputs. By applying the derivative on a vector $\hat{e}_k$ containing zeros and a one at index k, the kth column of the Jacobian is computed. Correspondingly, the kth row of J is computed by applying the transposed derivative on $\hat{e}_k$. In the special case $F : \mathbb{R}^N \to \mathbb{R}$ mapping to a scalar, the Jacobian reduces to a $1 \times N$-matrix containing the gradient of F which can be computed by applying $\mathrm{D}F|_{\mathbf{x}}^T$ on the scalar one, that is,

$$\nabla F = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \dots & \frac{\partial F_1}{\partial x_N} \end{pmatrix}^T = J^T = \mathrm{D}F^T\left(1\right). \tag{4}$$

Gradients are usually computed by the transposed derivative since this only requires one application of $\mathrm{D}F|_{\mathbf{x}}^T$ instead of N applications of $\mathrm{D}F|_{\mathbf{x}}$ to compute each column of J independently. As depicted in Figure 2A, `nlops` can have multiple inputs and outputs and there is a derivative for each combination of input and output. The derivatives are always evaluated at the inputs of the last call of F and there is a shared data structure to communicate this information. For example, the multiplication operator $F(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1\mathbf{x}_2$ stores $\mathbf{x}_2$ (and $\mathbf{x}_1$) needed by the derivative $\mathrm{D}_{\mathbf{x}_1}F|_{\mathbf{x}_1,\mathbf{x}_2} : \mathrm{d}\mathbf{x}_1 \mapsto \mathbf{x}_2\mathrm{d}\mathbf{x}_1$.

Composing operators

The crucial part of automatic differentiation is the possibility to *chain* `nlops` and compute the chained derivatives. Figure 2B shows the *chain* $H = G{\circ}F$ with its derivative $\mathrm{D}H|_{\mathbf{x}} = \mathrm{D}G|_{F(\mathbf{x})}{\circ}\mathrm{D}F|_{\mathbf{x}}$. As G is applied on $F(\mathbf{x})$, the derivative $\mathrm{D}G$ is automatically evaluated at $F(\mathbf{x})$. To compute the transposed $\mathrm{D}H|_{\mathbf{x}}^T = \mathrm{D}F|_{\mathbf{x}}^T{\circ}\mathrm{D}G|_{F(\mathbf{x})}^T$, the $\mathrm{D}F^T$ and $\mathrm{D}G^T$ are applied in reverse order, hence the name backpropagation. Similar to the *chain*, BART provides a set of functions for composing `nlops` with multiple inputs and outputs. These functions can be used to *combine* two `nlops` to one, to *link* an output of an `nlop` into one of its inputs, and to

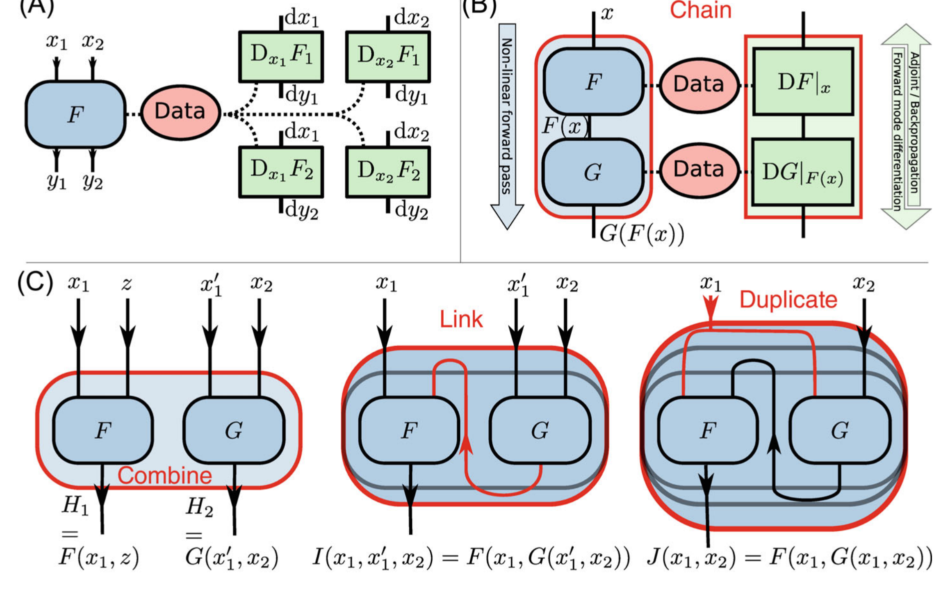


FIGURE 2 Basic concepts of `nlops`. (A) An atomic `nlop` exemplary with two complex-valued inputs $(\mathbf{x}_1, \mathbf{x}_2)$ and two outputs $(\mathbf{y}_1 = F_1(\mathbf{x}_1, \mathbf{x}_2), \mathbf{y}_2 = F_2(\mathbf{x}_1, \mathbf{x}_2))$ consisting of the forward `operator` F and its derivatives D_iF_o modeled by `linops`. F and D_iF_o communicate via a shared data structure. (B) Chaining of two `nlops` F and G. Since G is applied on the output $F(\mathbf{x})$, its derivative $\mathrm{D}G|_{F(\mathbf{x})}$ is automatically evaluated at $F(\mathbf{x})$. (C) The two `nlops` F and G are combined to form H, whose output 1 is linked into input 1 to form I, whose inputs 0 and 1 are duplicated to construct $J(\mathbf{x}_1, \mathbf{x}_2) = F(\mathbf{x}_1, G(\mathbf{x}_1, \mathbf{x}_2))$. The derivatives of the final operator are constructed automatically (not shown).

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

duplicate one of its inputs into another one. The action of these functions is presented in Figure 2C, where we demonstrate how two `nlops` F and G can be used to construct an `nlop` computing $J(\mathbf{x}_1, \mathbf{x}_2) = F(\mathbf{x}_1, G(\mathbf{x}_1, \mathbf{x}_2))$. The resulting `nlops` hold references to the base `nlops` to call them, and the derivatives are constructed automatically. Since *combine*, *duplicate* and *link* can be nested, generic compositions of `nlops` are possible.

Complex numbers

`nlops` in BART work with complex numbers, that is, single precision complex floats. The automatic differentiation framework is extended to complex numbers by identifying $\mathbb{C} \sim \mathbb{R}^2$. For example, a univariate complex mapping $F : x \mapsto y = F(x)$ is represented by

$$F : \begin{pmatrix} x_r \\ x_i \end{pmatrix} \mapsto \begin{pmatrix} y_r \\ y_i \end{pmatrix} \quad \mathrm{D}F : \begin{pmatrix} \mathrm{d}x_r \\ \mathrm{d}x_i \end{pmatrix} \mapsto \begin{pmatrix} \frac{\partial y_r}{\partial x_r} & \frac{\partial y_r}{\partial x_i} \\ \frac{\partial y_i}{\partial x_r} & \frac{\partial y_i}{\partial x_i} \end{pmatrix} \begin{pmatrix} \mathrm{d}x_r \\ \mathrm{d}x_i \end{pmatrix}. \tag{5}$$

This approach is equivalent to the so-called Wirtinger[26] or $\mathbb{CR}$-calculus[27] which introduces the complex derivatives $\frac{\partial y}{\partial x}$ and $\frac{\partial y}{\partial \bar{x}}$ to reformulate DF as

$$\begin{aligned} \mathrm{D}F : \mathrm{d}x &\mapsto \frac{\partial y}{\partial x}\mathrm{d}x + \frac{\partial y}{\partial \bar{x}}\overline{\mathrm{d}x} \quad \text{with} \quad \frac{\partial y}{\partial x} = \frac{1}{2}\left(\frac{\partial y}{\partial x_r} - i\frac{\partial y}{\partial x_i}\right) \\ \mathrm{D}F^T : \mathrm{d}y &\mapsto \overline{\frac{\partial y}{\partial x}}\mathrm{d}y + \frac{\partial y}{\partial \bar{x}}\overline{\mathrm{d}y} \qquad\qquad \frac{\partial y}{\partial \bar{x}} = \frac{1}{2}\left(\frac{\partial y}{\partial x_r} + i\frac{\partial y}{\partial x_i}\right). \end{aligned} \tag{6}$$

If F is holomorphic, $\frac{\partial y}{\partial \bar{x}} = 0$ holds, such that DF corresponds to the multiplication with the complex derivative $\frac{\partial y}{\partial x}$ and DF^T to the multiplication with its complex conjugate. Similarly, in the multivariate case $F : \mathbb{C}^N \to \mathbb{C}^M$, the derivative $\mathrm{D}F : \mathbb{C}^N \to \mathbb{C}^M$ is linear with respect to $\mathbb{C}$ iff F is holomorphic. If $\mathrm{D}F : \mathbb{C}^N \to \mathbb{C}^M$ is not linear with respect to $\mathbb{C}$, we still call the transposed of the real-valued derivative $\mathrm{D}F^T : \mathbb{R}^{2M} \to \mathbb{R}^{2N}$ the adjoint derivative $\mathrm{D}F^H : \mathbb{C}^M \to \mathbb{C}^N$.

The loss of a neural network in Eq. (1) must be real as there is no ordering on $\mathbb{C}$. In the picture of real-valued derivatives, training a network with complex-valued weights is equivalent to optimize the real and imaginary part of the weights independently. In the picture of Wirtinger calculus, we consider a mapping $F : \mathbb{C}^N \to \mathbb{R}$. Since the output is real, it holds $\overline{\frac{\partial F}{\partial x_j}} = \frac{\partial F}{\partial \bar{x}_j}$ such that

$$\begin{aligned} \mathrm{D}F^H(1) &= 2\begin{pmatrix} \frac{\partial F}{\partial \bar{x}_1} & \dots & \frac{\partial F}{\partial \bar{x}_N} \end{pmatrix}^T \\ &= \begin{pmatrix} \frac{\partial F_r}{\partial x_{1r}} + i\frac{\partial F_r}{\partial x_{1i}} & \dots & \frac{\partial F_r}{\partial x_{Nr}} + i\frac{\partial F_r}{\partial x_{Ni}} \end{pmatrix}^T. \end{aligned} \tag{7}$$

We stress the analogy to Eq. (4), that is, the real part of $\mathrm{D}F^H(1)$ is the gradient of F with respect to the real part of $\mathbf{x}$ and the imaginary part of $\mathrm{D}F^H(1)$ is the gradient of F with respect to the imaginary part of $\mathbf{x}$.

Implementation of operators

For interested programmers, we describe the C-implementation of `operators`, `linops`, and `nlops` in the nonlinear operator framework. `operators` are the basic structures of the framework. An `operator` holds an apply-function which is called when the operator is applied and a generic data structure which is passed to this function together with pointers to the input and output md-arrays (Figure 3A). An example for an `operator` is the chain-`operator` (Figure 3D), whose data structure holds references to the chained `operators` and whose apply function calls them one after another. A `linop` $A : \mathbb{C}^N \to \mathbb{C}^M$ models a linear operator by holding references to `operators` computing A and its adjoint A^H. For atomic, i.e. non-composed, `linops`, the `operators` have access to a shared data structure of type `linop_data_s` (Figure 3B). For example, a `linop` performing a matrix multiplication stores the matrix in this structure such that it can be accessed by the forward and adjoint `operators`. `linops` are chained by creating a new `linop` referring to the chained forward and adjoint `operators`. An `nlop` consists of an `operator` modeling the nonlinear forward operator and `linops` modeling the derivatives. For atomic `nlops`, the `linops` and the forward `operator`, have access to a shared data structure of type `nlop_data_s` (Figure 3C) to store the data necessary to evaluate the derivatives at the last input of the forward operator as described above. To implement a completely new `nlop`, the programmer needs to define the data structure `nlop_data_s` and functions to be called when the `nlop` or its (adjoint) derivative is applied. Other references and data structures are created automatically. All shared data structures use reference counting for automatic memory management (garbage collection). As `nlops` are implemented based on md-functions, they are automatically executed on the GPU if the inputs and outputs are located on the GPU.

Functional container

Generally, the execution properties of an `nlop` can be modified by encapsulating it in a container which itself is an `nlop`. We use such a container to implement checkpointing to reduce memory use. When the checkpointing-container is applied, the inputs are stored and the inner `nlop` is applied without saving data for computing its derivatives. When the derivatives are needed, the inner `nlop` is applied again using the inputs stored in the container and the data needed for the derivatives is recomputed. Thus, checkpointing can reduce the memory

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

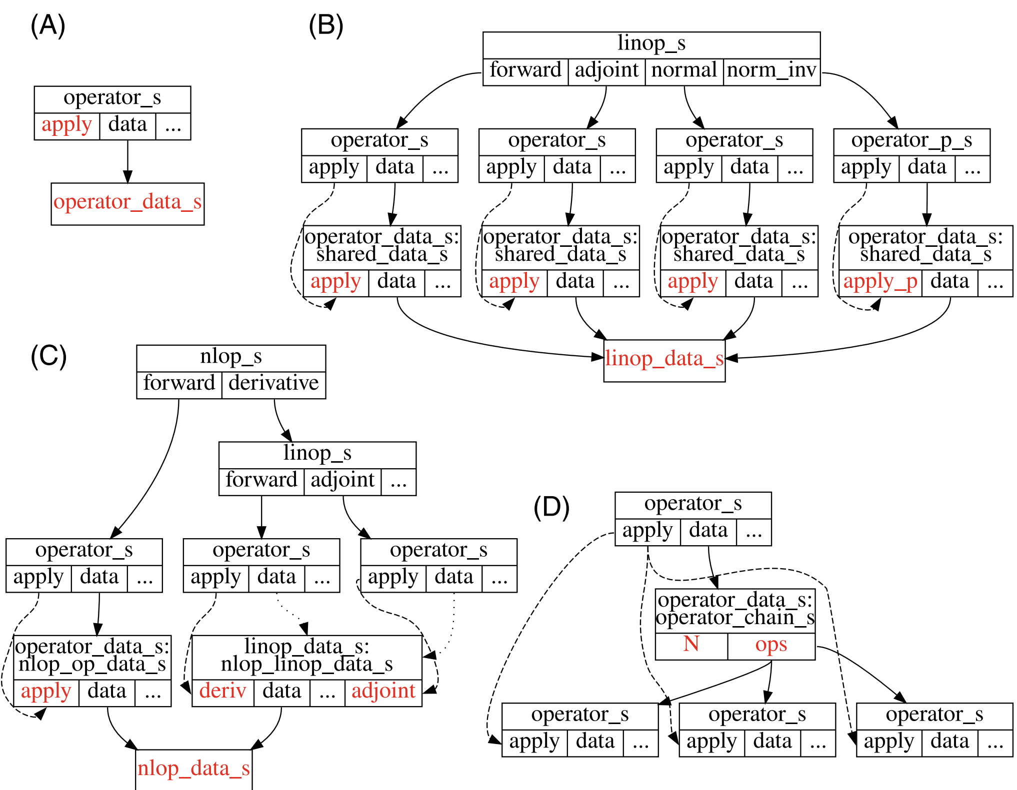


FIGURE 3 Schematic description of `operators`, `linops` and `nlops` as data structures in BART. Solid lines mean "points to," dotted lines "points to indirectly" and dashed lines "calls." Colons indicate specific realizations of a data structure, that is, `operator_chain_s` is the `operator_data_s` structure used for chaining operators. Objects required to create the respective structures are marked in red. Other structures and references are created automatically. (A) An `operator` holds a reference to a data structure and a function which is called when the `operator` is applied. (B) A `linop` holds references to multiple `operators` such as the forward and adjoint operator which share a common data structure. (C) An `nlop` holds references to the nonlinear forward `operator` and `linops` modeling the derivatives. The forward `operator` and `linops` have access to a shared data structure `nlop_data_s`. (D) The data structure of a chain-`operator` holds references to the chained `operators` which are applied sequentially, when the chain-`operator` is applied.

consumption at the price of multiple applications of the `nlop`.

Moreover, a functional container can be used to assign an `nlop` to a specific GPU. When such an nlop or its derivative is called, the CUDA context is switched to the selected GPU and all input data of the `nlop` are copied to the GPU. Afterward the inner `nlop` is called which uses the selected GPU for all its computation and memory allocations. By calling `nlops` assigned to different GPUs from different threads in parallel, we can efficiently distribute the memory and computation of the `nlops` to multiple GPUs (cf. Supporting Information Figure S2).

2.1.5 | Neural network library

The neural network (nn)-library contains our complex-valued implementations of typical operators used to construct neural networks, that is,

- fully connected (dense) layers
- (transposed / adjoint) convolutional layers
- dropout layers, max-pooling layers, batch normalization layer[28]
- activation layers: complex cardioid,[29] ℂReLU,[30,31] sigmoid and softmax
- loss functions: mean squared error (MSE), mean absolute difference, structural similarity index measure (SSIM),[32] generalized dice loss,[33] and categorical cross-entropy.

The corresponding `nlops` are implemented generically such that they act on N-dimensional complex-valued md-arrays and support operations (convolution, normalization, pooling) along arbitrary dimensions. However, currently convolutions are only backed by optimized GPU code for up to three dimensions. Moreover, the nn-library contains another wrapper for `nlops` to index the arguments (inputs and outputs) of `nlops` by meaningful names instead of numeric indices. The arguments are annotated by a type defining how the optimization algorithm treats this argument (weights, data, moving statistics of batch

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

normalization) and inputs corresponding to weights can be attached with an initializer and proximal operators for regularization.

Integration of TensorFlow graphs

The `nlop` framework also serves as a generic wrapper for computation graphs exported from other deep learning frameworks. As a proof of concept, we have implemented a wrapper for TensorFlow graphs based on the TensorFlow C API[1]. A pretrained neural network based on TensorFlow can be exported to a graph file which is imported by BART to construct an `nlop`. When this `nlop` is applied, the forward-pass of the TensorFlow graph is executed, while TensorFlow's gradients are used to compute the adjoint derivative of the `nlop`. The forward derivative for the TensorFlow wrapper is not implemented.

2.1.6 | Iterative training algorithms

Training a neural network corresponds to minimizing the loss $\sum_i L(\mathbf{y}_i, F(\mathbf{x}_i; \boldsymbol{\theta}))$ with respect to the weights $\boldsymbol{\theta}$ (cf. Eq. (1)). Having constructed an `nlop` representing F, we chain its output into another `nlop` L to generate a loss-`nlop`. This loss-`nlop` has two types of inputs, those corresponding to weights $\boldsymbol{\theta}$ and those corresponding to data $\mathbf{x}, \mathbf{y}$. The training dataset $\mathbf{x}_i, \mathbf{y}_i$ is split into mini-batches and in each iteration the weights $\boldsymbol{\theta}$ are updated based on the gradient with respect to these weights. For training neural networks, we have integrated incremental gradient methods such as stochastic gradient descent, Adam,[25] and iPALM[34] into BART's library for iterative algorithms.

2.2 | Applications and implemented networks

To demonstrate practicability of our framework, we have implemented and trained VarNet[17] and MoDL.[18] Both networks are motivated by unrolling an optimization algorithm solving the inverse problem

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{y}\|^2 + R(\mathbf{x}). \tag{8}$$

Here, $A = \mathcal{PFC}$ is the linear SENSE operator composed of the multiplication with the $\mathcal{C}$oil sensitivity maps, the $\mathcal{F}$ourier transform, and the projection to the sampling $\mathcal{P}$attern. $\mathbf{x}$ is the MR image to be reconstructed and $\mathbf{y}$ is the measured k-space data. $R(\mathbf{x})$ is a regularization term imposing prior knowledge on the reconstructed image $\mathbf{x}$.

[1] https://www.tensorflow.org/install/lang_c

We first describe the structure of both networks and our respective implementations. Afterwards, we describe how the TensorFlow wrapper can be used to integrate an externally trained regularizer $R(\mathbf{x})$ for reconstruction with BART. Scripts to reproduce training and application of the networks are available at https://github.com/mrirecon/deep-deep-learning-with-bart. To provide interested developers a starting point to implement neural networks in BART, we have implemented a toy network to classify handwritten digits of the MNIST[35] database. The network can be found in the BART source code at `src/mnist.c` and scripts to prepare the MNIST database are available in the script repository.

2.2.1 | Variational network

VarNet is motivated by solving Eq. (8) using an unrolled gradient descent algorithm that includes a trained regularizer R. The network is initialized with the adjoint reconstruction $\mathbf{x}^0 = A^H\mathbf{y}$ and updates the reconstruction $\mathbf{x}^t$ by

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \sum_{i=1}^{N_k} (K_i^t)^T \Phi_i^{t'} \left(K_i^t \mathbf{x}^t\right) - \lambda^t \left(A^H A\mathbf{x}^t - A^H\mathbf{y}\right)$$
$$0 \le t \le T-1. \tag{9}$$

Here, the network block $\sum_{i=1}^{N_k} (K_i^t)^T \Phi_i^{t'} \left(K_i^t \mathbf{x}^t\right)$ corresponds to the gradient of a regularizer $R^t(\mathbf{x}) = \sum_{i=1}^{N_k} \Phi_i^t(K_i^t \mathbf{x})$, where K is a convolution with N_k filters and Φ' is the derivative of a trainable activation function. The imaginary part of the convolved images $K_i^t\mathbf{x}$ is discarded to be consistent with the original implementation of VarNet. The last term corresponds to a gradient step of the data-consistency term $\|A\mathbf{x} - \mathbf{y}\|_2^2$ with trained step size λ^t. The BART implementation of VarNet can be trained and applied with the `reconet` command of the BART toolbox, that is (Listing 1),

```
$ bart reconet --network=varnet --train
<kspace> <coils> <weights> <reference>
$ bart reconet --network=varnet --apply
<kspace> <coils> <weights> <output>
```

Listing 1

`<kspace>`, `<coils>`, and `<reference>` are input files holding multidimensional arrays as training data or for inference. The data layout follows the BART convention and stacks independent datasets/volumes along the batch dimension 15. An undersampling pattern can be provided to subsample the k-space, otherwise the pattern is estimated from the k-space data. `<weights>` is a file

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

holding the network weights $\boldsymbol{\theta}$ which are an output of the training command and an input for the reconstruction/apply command. The network block is implemented to first reshape/transpose the current reconstruction $\mathbf{x}^t$ to the NDHWC data layout, where all BART dimensions not corresponding to the 3D-spatial coordinates (DHW) or the batch dimension (N) are interpreted as channel dimensions (C). In the second step, the network block is applied, and, finally, the result is reshaped/transposed back to the original layout. Further options, such as network parameters, training losses, initializations for the weights, or the training algorithm can be configured using command line options. The default hyperparameter are based on the TensorFlow implementation[2], that is, $T = 10$ iterations, $N_k = 24$ 11×11-convolution filter and $N_w = 31$ Gaussian radial basis functions to construct the activation Φ'. This results in 65 530 real-valued trainable parameters. iPALM is used as training algorithm. The `-normalize` option can be used to scale the data such that $1 = \max |\mathbf{x}^0|$. As our implementation is equivalent to the original one, weights trained with TensorFlow can be exported for inference with BART.

2.2.2 | MoDL

The MoDL[18] network is another unrolled network initialized with $\mathbf{x}^0 = A^H\mathbf{y}$. A residual network $\mathcal{D}_W$ denoises the current reconstruction and data-consistency is imposed by a proximal mapping. The iterations read

$$\begin{aligned} \mathbf{x}^{t+1} &= \underset{\mathbf{x}}{argmin} \left[\|A\mathbf{x} - \mathbf{y}\|^2 + \lambda \|\mathbf{x} - \mathcal{D}_W(\mathbf{x}^t)\|^2 \right] \\ &= \underbrace{\left(A^H A + \lambda \mathbb{1}\right)^{-1}}_{Q} \left(A^H\mathbf{y} + \lambda \mathcal{D}_W(\mathbf{x}^t)\right) \qquad 0 \le t \le T-1. \end{aligned} \tag{10}$$

The residual network $\mathcal{D}_W$ consists of L convolutional layers with F filters followed by batch normalization layers and ReLU activation functions. The `reconet` command with the `-network=modl` option is used to train MoDL. By default, the Adam algorithm and the hyperparameter from the TensorFlow implementation[3] are used, that is, the network is unrolled for $T = 10$ unrolled iterations with shared weights and each residual block contains $L = 5$ convolutional layers with $F_c = 32$ complex-valued filters (instead of $F_r = 64$ real-valued in the TensorFlow implementation). Thus, our implementation has 28 364 complex-valued (= 56 728 real-valued) trainable parameters in contrast to 112 001 real-valued parameters in the Tensor-Flow implementation. To compute $Q(\mathbf{x})$, we have implemented a generic inversion module for positive-definite self-adjoint linear operators $S_\lambda : \mathbb{C}^N \to \mathbb{C}^N$ parametrized with parameters $\lambda \in \mathbb{C}^M$. Given an `nlop` $\mathcal{S}$ that has λ as second input and applies the parametrized linear operator S_λ to its first input, we construct the `nlop` $\mathcal{S}^{-1}$ applying the inverse S_λ^{-1}. These `nlops` are defined by

$$\begin{aligned} &\mathcal{S} : \mathbb{C}^N \times \mathbb{C}^M \to \mathbb{C}^N && \mathcal{S}^{-1} : \mathbb{C}^N \times \mathbb{C}^M \to \mathbb{C}^N \\ &(\mathbf{x}, \lambda) \mapsto \mathbf{y} = S_\lambda \mathbf{x} && (\mathbf{y}, \lambda) \mapsto \mathbf{x} = S_\lambda^{-1}\mathbf{y}\,. \end{aligned} \tag{11}$$

Note that the derivatives of $\mathcal{S}^{-1}$ can be expressed in terms of $\mathcal{S}$ and its derivatives, that is,

$$\begin{aligned} &\mathrm{D}_{\mathbf{y}}\mathcal{S}^{-1}|_{\mathbf{y},\lambda} : \mathrm{d}\mathbf{y} \mapsto \mathrm{d}\mathbf{x} = S_\lambda^{-1}\mathrm{d}\mathbf{y} \\ &\mathrm{D}_\lambda\mathcal{S}^{-1}|_{\mathbf{y},\lambda} : \mathrm{d}\lambda \mapsto \mathrm{d}\mathbf{x} = -S_\lambda^{-1} \circ \mathrm{D}_\lambda \mathcal{S}|_{\mathcal{S}^{-1}(\mathbf{y},\lambda),\lambda}\mathrm{d}\lambda\,. \end{aligned} \tag{12}$$

As proposed by Aggarwal et al.[18] we use the conjugate gradient algorithm to apply S_λ^{-1}.

2.2.3 | Extensions to the SENSE-model

BART's implementation of the SENSE model is generic in the sense that it can handle multiple sets of coil sensitivity maps (soft-SENSE[36]) and supports non-Cartesian sampling patterns. The soft-SENSE model is suitable if the object exceeds the FOV since one set of coil sensitivity maps can not explain infolding artifacts.[36] In the context of deep-learning, the soft-SENSE model has been used recently.[37-39] VarNet and MoDL only update the image corresponding to the first set of coil sensitivity maps in the network block. We use the MSE loss on the coil images since the coil images serve as a reference independent of the estimated coil sensitivity maps.

Reconstruction networks for non-Cartesian sampling trajectories have been investigated recently.[37,40-42] BART implements the nonuniform (nu)FFT as a `linop` which is integrated in the SENSE model of VarNet and MoDL. To save expensive gridding steps of the nuFFT, we precompute the adjoint reconstruction $A^H\mathbf{y}$ and the point spread function (PSF) of the non-Cartesian sampling pattern $\mathcal{P}$ for the whole dataset. The joint forward-backward nuFFT $\mathcal{F}^H\mathcal{P}^H\mathcal{P}\mathcal{F}$ is implemented by the convolution with the PSF (Toeplitz trick),[43,44] which significantly speeds up computations on the GPU.[45,46] We initialize the non-Cartesian networks with a SENSE reconstruction $\mathbf{x}^0 = QA^H\mathbf{y}$. For MoDL, the number of CG-iterations in each data-consistency block has been increased from 10 to 30, while the number of unrolled iterations has been reduced to $T = 5$.

[2] https://github.com/VLOGroup/mri-variationalnetwork, Commit: 4b6855f

[3] https://github.com/hkaggarwal/modl, Commit: 428ef84

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

2.2.4 | Image reconstruction using a learned prior

An alternative approach of using neural networks for MRI reconstruction is learning prior knowledge about the image distribution by learning a regularizer $R(\mathbf{x})$ independently of the reconstruction. For reconstruction, the learned regularizer is inserted into Eq. (8). One approach to learn a regularizer is based on deep Bayesian estimation.[14] The resulting regularizer is given by

$$R(\mathbf{x}) = -\lambda \log p(\mathbf{x}; \mathtt{Net}(\mathbf{x}, \boldsymbol{\theta}^*)). \tag{13}$$

Here, $\mathtt{Net}(\mathbf{x}, \boldsymbol{\theta}^*)$ denotes the PixelCNN++[47] which is trained to predict the conditioned distribution parameters of the mixture of logistic distributions that are used to model the image distribution. Inserting the regularizer R defined in Eq. (13) into the optimization problem Eq. (8) corresponds to a maximum a posteriori estimation for the reconstructed image. For more details, we refer to the original publication.[14]

For reconstruction, the trained TensorFlow graph computing $R(\mathbf{x})$ is exported and loaded into BART using the TensorFlow wrapper described above. The resulting `nlop` is used to construct the corresponding proximal operator

$$\mathrm{prox}_R(\mathbf{v}) = \underset{\mathbf{x}}{argmin} \frac{1}{2} \|x - v\|^2 + R(\mathbf{x}) \ . \tag{14}$$

The proximal operator is computed by the gradient descent algorithm using automatic differentiation to compute ∇R. The proximal operator can be plugged into any of BART's proximal operator based iterative optimization algorithms using the `pics` command, that is (Listing 2),

```
$ bart pics -R TF:{model_path}:lambda
<kspace> <coils> <output>
```

Listing 2

3 | RESULTS

3.1 | Reconstructions with BART

We have trained VarNet and MoDL on the datasets provided with the respective publications using the BART and TensorFlow implementations to compare the reconstruction quality. VarNet was trained for 30 epochs algorithm with a batch size $N_b = 10$ on 300 randomly ordered slices of 15 subjects from the `coronal_pd_fs` directory, while 20 slices of the remaining five subjects were used for evaluation. The 15-coil fully sampled k-space data was retrospectively subsampled (4-fold regular undersampling, 28 auto calibration lines). MoDL was trained with batch size $N_b = 10$ in a two-step-approach, that is, the weights were initialized for 100 epochs using $T = 1$ unrolled iteration and afterwards the network was trained for 50 epochs with $T = 10$. The brain dataset of MoDL consists of 5 subjects acquired with a 12 channel head coil. 90 slices of the first 4 subjects (360 in total) were used for training and 100 slices of the remaining subject for testing. Subsampled k-space data was generated from the fully sampled images by multiplying them with provided coil sensitivity maps, Fourier transform, subsampling (variable density with acceleration 6, no auto-calibration region) and addition of Gaussian noise with SD $\sigma = 0.001$. This procedure is used in the TensorFlow implementation to produce training data. We only used normalization as described above for the knee data of VarNet. We show example reconstructions based on the respective networks and implementations in Figure 4. Both implementations of the respective networks perform quite similar and better than the classical ℓ_1-Wavelet regularized reconstruction. To support this statement quantitatively, we computed the PSNR and SSIM for each slice in the evaluation dataset and visualize the results in the boxplots in Figure 4. Moreover, we compare in Supporting Information Figure S1 the mean PSNR and SSIM of the VarNet evaluation dataset computed after each training epoch.

To demonstrate the benefits of the soft-SENSE model in the case that the object exceeds the FOV, we simulated k-space data with a reduced FOV from the fully sampled knee dataset and trained VarNet and MoDL on this dataset. ESPIRiT[36] was used to estimate either one or two sets of coil sensitivity maps from the simulated k-space. Respective example reconstructions and quantitative metrics computed on the evaluation dataset are presented in Figure 5. Reconstructions using two sets of coil sensitivity maps show less aliasing artifacts and are superior in terms of SSIM and PSNR.

Further, we used the knee dataset to simulate non-Cartesian k-space data (radial trajectory with 44 spokes) and trained the network on this simulated data. In Figure 6, we present an example reconstruction using the non-Cartesian versions of VarNet and MoDL. The learned methods improve the image quality compared to the classical ℓ_1-Wavelet regularized reconstruction slightly.

3.2 | Computational performance

We compared the BART and the TensorFlow[4] implementations of MoDL and VarNet in terms of training

[4]VarNet: https://github.com/VLOGroup/mri-variationalnetwork, Commit: 653630b; MoDL: https://github.com/hkaggarwal/modl, Commit: 428ef84.

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

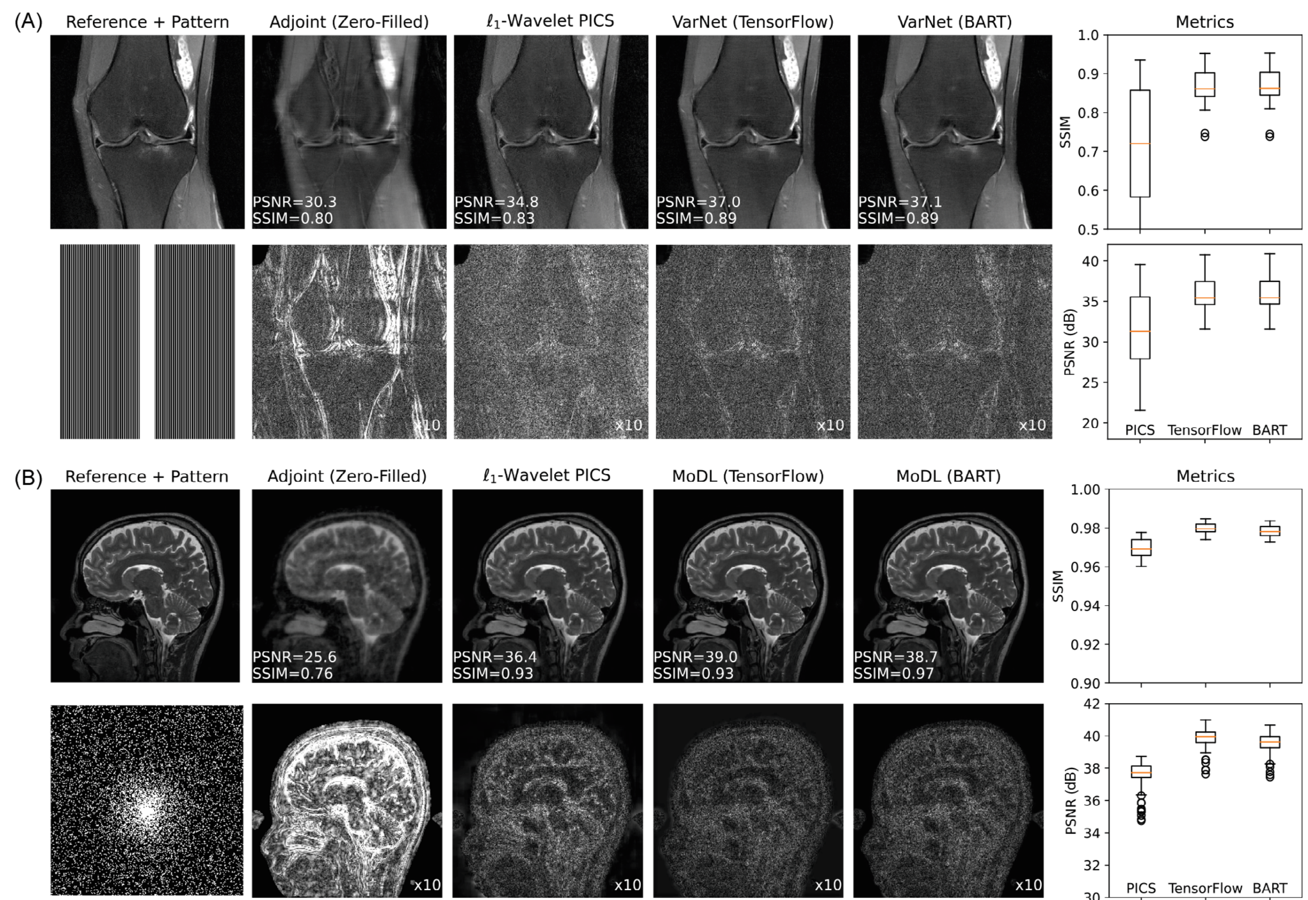


FIGURE 4 Comparison of the TensorFlow and BART implementation of VarNet (A) and MoDL (B). For reference, we also show the results of the adjoint reconstruction $A^H\mathbf{y}$ and an ℓ_1-Wavelet regularized SENSE reconstruction computed using the BART pics tool. Boxplots are based on PSNR and SSIM of the respective evaluation datasets using the coil sensitivities as foreground mask. This mask explains the discrepancy to the SSIM values given at the reconstructed images.

time, inference time, and memory consumption on four different Nvidia GPUs, that is, A100-SXM-80GB, Tesla V100-SXM2-32G, TITAN Xp (12 GB) and GTX TITAN X (12 GB). We use TensorFlow 1.15 maintained by NVIDIA to support current GPUs[5] with cuBLAS 11.7, and cuFFT 10.6, cuDNN 8.3. For VarNet we also experimented with the implementation[6] based on TensorFlow-ICG[7]. The training time of BART was measured in two settings: once with CUDA 11.2, cuDNN 8.3 and the use of nondeterministic algorithm and once without cuDNN and only using deterministic algorithms. All network parameters were chosen as described before except the number of unrolled iterations of MoDL which was reduced to $T = 5$ to fit in 12 GB GPU memory. For MoDL, only the time for the second part of the two-step training has been measured. The results are presented in Figure 7. In general, the computation time of the BART and TensorFlow implementations are comparable, however, TensorFlow performs better on the older GeForce GTX TITAN X GPU. BART's implementation of VarNet is distributed to two GPUs by stacking two versions of the network each associated to the respective GPU along the batch dimension (cf. Supporting Information Figure S2). The overhead due to multi-GPU synchronization is minimal resulting in a training time reduced by 47%–49% depending on the GPU. Batch-normalization used by MoDL requires inter-batch synchronization such that only the data-consistency blocks are distributed to multiple GPUs reducing the benefit of multiple GPUs. The inference time was measured on the respective evaluation datasets. To reduce the bias due to different preprocessing procedures on the CPU in the respective implementations, we also measured the total execution time of GPU kernels using NVIDIA Nsight Systems. In general, the BART implementations achieve similar performance to the TensorFlow implementations.

We measured the peak GPU memory allocation during training and inference for the respective implementations

[5]https://github.com/NVIDIA/tensorflow

[6]https://github.com/VLOGroup/mri-variationalnetwork, Commit: 4b6855f

[7]https://github.com/VLOGroup/tensorflow-icg; CUDA 8.0; cuDNN 7.0

FIGURE 5 Comparison of two example reconstructions with MoDL and variational network (VarNet) using one set of coil sensitivity maps (usual SENSE) and two sets of coil sensitivity maps (soft-SENSE). The aliased k-space data is simulated by first zero-padding the fully-sampled coil-images and afterwards subsampling the k-space by a factor of two before applying the usual sampling pattern (every fourth line and 28 auto calibration lines). The usage of two sets of coil sensitivity maps reduce undersampling artifacts (cf. arrows) and improves the PSNR and SSIM for VarNet and MoDL.

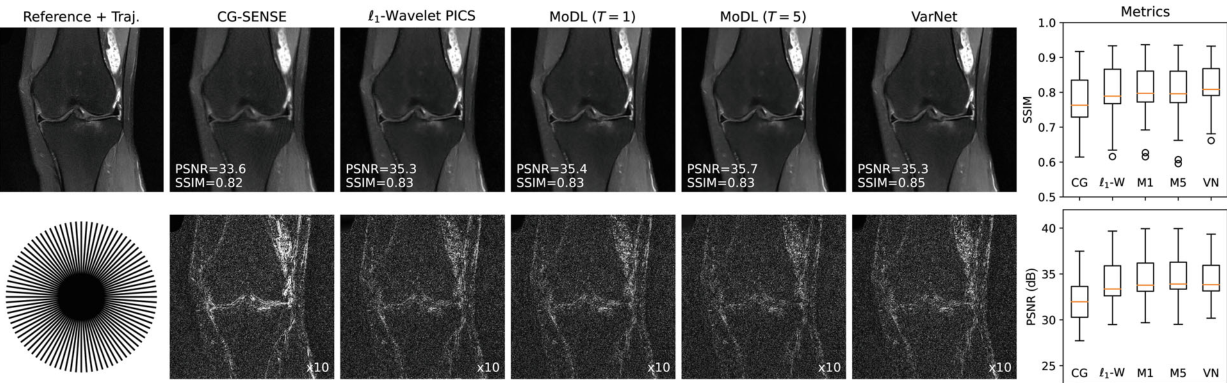


FIGURE 6 Comparison of MoDL and variational network (VarNet) for non-Cartesian reconstructions using a radial trajectory with 44 spokes. The fully sampled k-space data from the reference knee image in Figure 4 was interpolated on the trajectory to simulate the non-Cartesian k-space data. For reference, we show the results of the adjoint reconstruction $A^H\mathbf{y}$ with density compensation, a CG-SENSE and ℓ_1-Wavelet regularized reconstruction computed using the BART pics tool.

of MoDL and VarNet and present the results in Table 1. Since allocating GPU memory is expensive, both, BART and TensorFlow, use a memory cache to reuse allocated memory. While BART allocates memory on demand, TensorFlow pre-allocates larger memory blocks such that the peak memory allocation exceeds the actually required memory. Thus, we also state the memory allocation before reaching the peak allocation which serves as a loose lower bound. The memory needed for training and inference is overall similar across implementations. The TensorFlow implementation of VarNet computes the gradient steps in the data-consistency block without removing frequency oversampling which is a possible reason for the higher memory requirement. Results on the TITAN Xp are based on an old version of BART since the GPU broke during the revision.

3.3 | Image reconstruction using a learned prior

In Figure 8, we present the reconstruction based on a learned prior. The prior was retrained on the brain dataset described in Reference 14. Data for reconstruction was

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

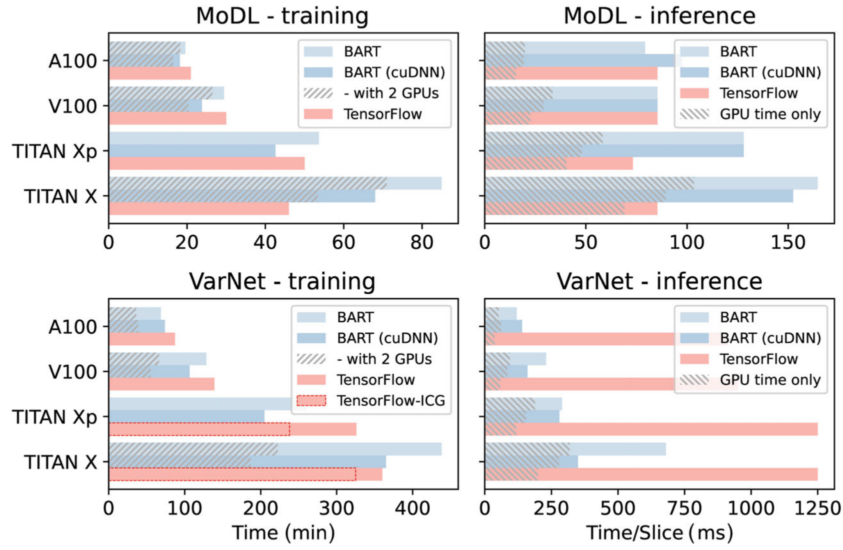
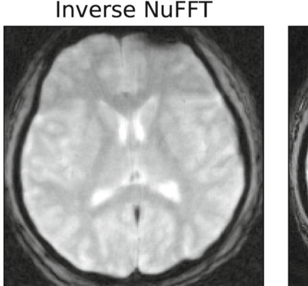


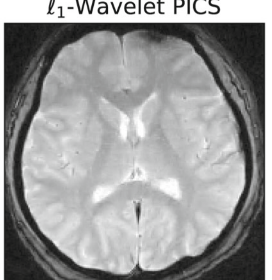
FIGURE 7 Comparison of training (left) and inference (right) time for MoDL and VarNet on different GPUs (full names in text). We observed slow host to device copies on the TITAN Xp which might affect the TensorFlow result of MoDL on this GPU. In general, the BART and TensorFlow implementations provide similar performance.

TABLE 1 GPU-Memory (in GB) used by BART and TensorFlow to train/infer MoDL and variational network (VarNet) on different GPUs (full names in text)

	Training		Inference	
MoDL	**BART**	**TensorFlow**	**BART**	**TensorFlow**
A100	8.9 (9.6)	10.2 (5.8)	1.8 (2.2)	1.9 (1.4)
V100	8.6 (8.9)	9.6 (2.9)	1.6 (1.7)	1.4 (1.1)
TITAN Xp	11.8 (12.6)	9.2 (4.9)	1.7 (1.8)	4.0 (0.7)
TITAN X	8.3 (9.0)	9.2 (4.9)	1.1 (1.1)	4.0 (0.6)
VarNet	**BART**	**TensorFlow**	**BART**	**TensorFlow**
A100	6.6 (6.8)	18.7 (10.2)	1.6 (2.0)	1.9 (1.7)
V100	6.2 (6.3)	18.2 (9.6)	1.3 (1.6)	1.4 (1.1)
TITAN Xp	6.3 (6.3)	12.4 (9.2)	1.2 (1.4)	1.1 (0.8)
TITAN X	5.8 (5.8)	12.4 (9.2)	1.0 (1.0)	1.0 (0.5)

Note: In parentheses, we provide for BART the memory if cuDNN is used and for TensorFlow the memory used before the peak-allocation is reached, which serves as a loose lower bound.





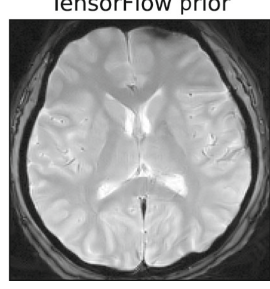


FIGURE 8 Brain images reconstructed from 60 radial k-space spokes via a coil-combined inverse nuFFT, an ℓ_1-Wavelet regularized PICS reconstructions, and a PICS reconstruction using a learned log-likelihood prior (left to right)

acquired on a Siemens Skyra 3T scanner (Siemens Healthcare GmbH). For reconstruction, we used 60 spokes of a radial Flash sequence (TR = 770 ms, TE = 16ms, FA = 18°). The coil sensitivity maps were estimated using ESPIRiT and gradient delays were corrected based on RING.[48] We compare a reconstruction using the inverse nuFFT, an ℓ_1-Wavelet regularized PICS reconstruction and a reconstruction using the learned log-likelihood prior (cf. Eq. (13)). The learned log-likelihood prior results in an improved reconstruction compared to the classical ℓ_1-Wavelet regularized reconstruction.

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

4 | DISCUSSION

In this work, we describe a framework for deep learning that was included in the BART toolbox. The framework is based on an extension of the existing nonlinear operator framework in BART that provides automatic differentiation and directly integrates BART's existing MRI-specific operators such as multidimensional FFT, nuFFT, and SENSE operators and complements it with many operators commonly used to construct neural networks. A sophisticated framework for constructing complex neural networks was added. We also implemented various optimizing techniques and achieve computational performance similar to other deep-learning frameworks. Distributing computation to multiple GPUs can reduce computation time further. Finally, we added new optimization algorithms such as stochastic gradient descent, iPALM, and Adam, which are popular for training neural networks. To demonstrate practicality of the framework, we implemented and trained the VarNet and MoDL in BART. Our implementation achieves similar performance in terms of reconstruction quality and training time compared to the original implementations based on TensorFlow. Further, BART's generic formulation of the SENSE model including the non-Cartesian and soft-SENSE formulation together with a flexible parametrization of the training procedure in terms of training algorithm, loss functions, and training target (coil combined reconstruction, RSS reconstruction or coil images) enables direct use of VarNet and MoDL for many applications.

State-of-the-art deep-learning-based MR image reconstruction algorithms combine two fields of research, that is, the field of machine learning and the field of classical MRI reconstruction methods. For both fields, mature software frameworks/toolboxes already exist. Hence, various approaches exist to develop algorithms combining both fields, the two extreme cases are (1) MRI-specific operations can be re-implemented in deep-learning frameworks and (2) neural networks can be re-implemented in MRI frameworks. Both approaches have different advantages and disadvantages. Deep learning frameworks such as TensorFlow or PyTorch are driven by large communities and recent developments in the field of deep learning are quickly integrated. Moreover, many tutorials based on standard frameworks exist and simple scripting based on Python reduces the barrier to entry. The frameworks are designed for large-scale datasets and support most recent hardware as well as direct integration into cloud solutions of various providers. However, all these features come at a price: Current deep learning libraries use many external libraries with complex dependencies. Updating some libraries in the backend or the framework itself might produce version conflicts which are hard to resolve. Long-term reproducibility of research results is difficult to achieve in this environment. One solution is to freeze the environment in a software container which contains the specific software versions that are known to work. In this way, containers can facilitate the reproduction of results and the translation of working setups to clinical pipelines.[49,50] While freezing setups is a legitimate approach for production environments or reproducing results, it is not a sustainable solution for long-term research and development, where new developments need to build on top of existing code.

On the other side, BART is designed for rapid prototyping, reproducible research and clinical translation. It depends only on a few external libraries such as FFTW, BLAS implementations or—if compiled with GPU support—CUDA, making it simple to integrate into different software environments. Where standard deep learning frameworks benefit from large community support integrating new deep learning features, BART benefits from years of research on MR image reconstruction. For example, a crucial part of most multicoil reconstruction networks is the estimation of the coil sensitivity maps in a preprocessing step. Since BART implements several calibration methods such as ESPIRiT or NLINV, a full reconstruction pipeline based on VarNet or MoDL can be implemented completely in BART. Advanced concepts from MRI implemented in BART can be directly used in these machine learning methods. For example, our data consistency modules are implemented using a generalized SENSE model supporting multiple sets of coil sensitivity maps, non-Cartesian trajectories and higher dimensional extensions which have been shown beneficial for dynamic MRI.[51,52] Concerning performance of training neural networks, we have demonstrated that BART can compete with the TensorFlow implementation of MoDL and VarNet. We hope that the deep integration of MRI-specific operators will be appreciated by researchers from other groups such that they will contribute to this open-source deep-learning framework. Further, we plan to reduce the entry barrier for possible users by extending the TensorFlow wrapper such that it can be used for defining denoising networks which can then be combined with BART's data-consistency modules in the `reconet` command. BART development makes uses of a continuous integration framework that uses automatic testing. Based on this, we aim for long-term reproducibility of published results even with new BART versions. Finally, BART is already widely used in the community of MRI research and also used for clinical research as part of automatic reconstruction frameworks such as Gadgetron[49,53] or Yarra.[54] Thus, we believe that the integration of neural networks into BART will also facilitate research and clinical translation of deep learning methods for image reconstruction.

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

5 | CONCLUSION

By integrating a complete set of tools for training and using neural networks into BART, we provide a general framework for research in image reconstruction that combines state-of-the-art methods for image reconstruction with deep-learning-based methods. The implementation of two recent deep-learning-based methods in BART demonstrates similar performance as their original TensorFlow-based implementations.

ACKNOWLEDGEMENTS

This work was funded by the “Niedersächsisches Vorab” funding line of the Volkswagen Foundation, and funded in part by NIH under grant U24EB029240 and funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant UE 189/1-1, under Germany's Excellence Strategy - EXC 2067/1- 390729940, and with Project-ID 432680300 - SFB 1456. This work was supported by the DZHK (German Centre for Cardiovascular Research). We gratefully acknowledge the support of the NVIDIA Corporation with the donation of one NVIDIA TITAN Xp GPU for this research. Open Access funding enabled and organized by Projekt DEAL.

CONFLICT OF INTEREST

The authors declare no competing interests.

DATA AVAILABILITY STATEMENT

In the spirit of reproducible research, code to reproduce the experiments are available on https://github.com/mrirecon/deep-deep-learning-with-bart. BART itself is available on https://github.com/mrirecon/bart. The data that support the findings of this study are available from the publications of the Variational Network [1] and MoDL [2]. (Converted) Data are available at http://dx.doi.org/10.5281/zenodo.6482960 (VarNet) and http://dx.doi.org/10.5281/zenodo.6481291 (MoDL).

ORCID

Moritz Blumenthal https://orcid.org/0000-0002-2127-8365
Guanxiong Luo https://orcid.org/0000-0001-8005-4639
H. Christian M. Holme https://orcid.org/0000-0002-8619-0444
Martin Uecker https://orcid.org/0000-0002-8850-809X

REFERENCES

1. Sodickson DK, Manning WJ. Simultaneous acquisition of spatial harmonics (SMASH): fast imaging with radiofrequency coil arrays. *Magn Reson Med*. 1997;38:591-603.
2. Griswold MA, Jakob PM, Heidemann RM, et al. Generalized autocalibrating partially parallel acquisitions (GRAPPA). *Magn Reson Med*. 2002;47:1202-1210.
3. Lustig M, Pauly JM. SPIRiT: iterative self-consistent parallel imaging reconstruction from arbitrary k-space. *Magn Reson Med*. 2010;64:457-471.
4. Pruessmann KP, Weiger M, Scheidegger MB, Boesiger P. SENSE: sensitivity encoding for fast MRI. *Magn Reson Med*. 1999;42:952-962.
5. Lustig M, Donoho D, Pauly JM. Sparse MRI: the application of compressed sensing for rapid MR imaging. *Magn Reson Med*. 2007;58:1182-1195.
6. Block KT, Uecker M, Frahm J. Undersampled radial MRI with multiple coils. Iterative image reconstruction using a total variation constraint. *Magn Reson Med*. 2007;57:1086-1098.
7. Liang D, Liu B, Wang JJ, Ying L. Accelerating SENSE using compressed sensing. *Magn Reson Med*. 2009;62:1574-1584.
8. Abadi M, Barham P, Chen J, et al. TensorFlow: a system for large-scale machine learning. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16); 2016:265-283; Savannah, Georgia.
9. Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*. Vol 32. Vancouver; 2019:8024-8035.
10. Epperson KS, Sawyer MA, Lustig M, et al. Creation of an MRI data repository for validating compressed sensing and other acceleration techniques. SMRT 22nd Annual Meeting, Salt Lake City, 2013;22:1.
11. Ong F, Amin S, Vasanawala S, Lustig M. mridata.org: an open archive for sharing MRI raw data. *Proc Intl Soc Mag Reson Med*. 2018;26:3425.
12. Knoll F, Murrell T, Sriram A, et al. Advancing machine learning for MR image reconstruction with an open competition: Overview of the 2019 fastMRI challenge. *Magn Reson Med*. 2020;84:3054-3070.
13. Zhu B, Liu JZ, Cauley SF, Rosen BR, Rosen MS. Image reconstruction by domain-transform manifold learning. *Nature*. 2018;555:487-492.
14. Luo G, Zhao N, Jiang W, Hui ES, Cao P. MRI reconstruction using deep Bayesian estimation. *Magn Reson Med*. 2020;84:2246-2261.
15. Yang G, Yu S, Dong H, et al. DAGAN: deep de-aliasing generative adversarial networks for fast compressed sensing MRI reconstruction. *IEEE Trans Med Imag*. 2018;37:1310-1321.
16. Kofler A, Haltmeier M, Schaeffter T, et al. Neural networks-based regularization for large-scale medical image reconstruction. *Phys Med Biol*. 2020;65:135003.
17. Hammernik K, Klatzer T, Kobler E, et al. Learning a variational network for reconstruction of accelerated MRI data. *Magn Reson Med*. 2017;79:3055-3071.
18. Aggarwal HK, Mani MP, Jacob M. MoDL: model-based deep learning architecture for inverse problems. *IEEE Trans Med Imaging*. 2019;38:394-405.
19. Schlemper J, Caballero J, Hajnal JV, Price AN, Rueckert D. A deep cascade of convolutional neural networks for dynamic mr image reconstruction. *IEEE Trans Med Imag*. 2018;37:491-503.
20. Uecker M, Ong F, Tamir JI, et al. Berkeley advanced reconstruction toolbox. *Proc Intl Soc Mag Reson Med*. 2015;23:2486.
21. Holme HCM, Rosenzweig, Ong F, et al. ENLIVE: an efficient nonlinear method for calibrationless and robust parallel imaging. *Sci Rep*. 2019;9:3034.

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

22. Wang X, Tan Z, Scholand N, Roeloffs V, Uecker M. Physics-based reconstruction methods for magnetic resonance imaging. *Philos Trans R Soc A*. 2021;379:20200196.
23. Blumenthal M, Uecker M. Deep deep learning with BART. *Proc Intl Soc Mag Reson Med*. 2021;29:1754.
24. Luo G, Blumenthal M, Uecker M. Using data-driven image priors for image reconstruction with BART. *Proc Intl Soc Mag Reson Med*. 2021;29:3768.
25. Kingma DP, Ba J. Adam: a method for stochastic optimization. arXiv. 2014;arXiv:1412.6980.
26. Wirtinger W. Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen. *Math Ann*. 1927;97:357-375.
27. Kreutz-Delgado K. The complex gradient operator and the CR-calculus. arXiv. arXiv:0906.4835, 2009.
28. Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. Proceedings of the 32nd International Conference on Machine Learning; Vol. 37, 2015:448-456; Lille, France.
29. Virtue P, Yu SX, Lustig M. Better than real: complex-valued neural nets for MRI fingerprinting. Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP); 2017; Beijing.
30. Trabelsi C, Bilaniuk O, Zhang Y, et al. Deep complex networks. Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Conference Track Proceedings; 2018; Vancouver, British Columbia, Canada.
31. Cole E, Cheng J, Pauly J, Vasanawala S. Analysis of deep complex-valued convolutional neural networks for MRI reconstruction and phase-focused applications. *Magn Reson Med*. 2021;86:1093-1109.
32. Zhao H, Gallo O, Frosio I, Kautz J. Loss functions for image restoration with neural networks. *IEEE Trans Comput Imag*. 2017;3:47-57.
33. Sudre CH, Li W, Vercauteren T, Ourselin S, Cardoso MJ. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer International Publishing; 2017: 240-248.
34. Pock T, Sabach S. Inertial proximal alternating linearized minimization (iPALM) for nonconvex and nonsmooth problems. *SIAM J Img Sci*. 2016;9:1756-1787.
35. LeCunn Y, Cortes C. The MNIST database of handwritten digits http://yann.lecun.com/exdb/mnist/.
36. Uecker M, Lai P, Murphy MJ, et al. ESPIRiT—An eigenvalue approach to autocalibrating parallel MRI: where SENSE meets GRAPPA. *Magn Reson Med*. 2014;71:990-1001.
37. Sandino CM, Lai P, Vasanawala SS, Cheng JY. Accelerating cardiac cine MRI using a deep learning-based ESPIRiT reconstruction. *Magn Reson Med*. 2021;85:152-167.
38. Hammernik K, Schlemper J, Qin C, Duan J, Summers RM, Rueckert D. Systematic evaluation of iterative deep neural networks for fast parallel MRI reconstruction with sensitivity-weighted coil combination. *Magn Reson Med*. 2021;86:1859-1872.
39. Johnson PM, Tong A, Donthireddy A, et al. Deep learning reconstruction enables highly accelerated biparametric MR imaging of the prostate. *J Magn Reson Imaging*. 2022;56:184-195.
40. Schlemper J, Mohseni Salehi SS, Kundu P, et al. Nonuniform variational network: deep learning for accelerated nonuniform MR Image reconstruction. *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*. Springer; 2019:57-64.
41. Kofler A, Haltmeier M, Schaeftter T, Kolbitsch C. An end-to-end-trainable iterative network architecture for accelerated radial multi-coil 2D cine MR image reconstruction. *Med Phys*. 2021;48:2412-2425.
42. Ramzi Z, Starck JL, Ciuciu P. Density compensated unrolled networks for non-cartesian mri reconstruction. Proceedings of the 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI); 2021; Nice, France.
43. Wajer FTAW, Pruessmann KP. Major speedup of reconstruction for sensitivity encoding with arbitrary trajectories. *Proc Intl Soc Mag Reson Med*. 2001;9:0767.
44. Fessler JA, Lee S, Olafsson VT, Shi HR, Noll DC. Toeplitz-based iterative image reconstruction for MRI with correction for magnetic field inhomogeneity. *IEEE Trans Signal Process*. 2005;53:3393-3402.
45. Uecker M, Zhang S, Frahm J. Nonlinear inverse reconstruction for real-time MRI of the human heart using undersampled radial FLASH. *Magn Reson Med*. 2010;63:1456-1462.
46. Baron CA, Dwork N, Pauly JM, Nishimura DG. Rapid compressed sensing reconstruction of 3D non-Cartesian MRI. *Magn Reson Med*. 2017;79:2685-2692.
47. Salimans T, Karpathy A, Chen X, Kingma DP. PixelCNN++: improving the PixelCNN with discretized logistic mixture likelihood and other modifications. *arXiv*. 2017;arXiv:1701.05517.
48. Rosenzweig S, Holme HCM, Uecker M. Simple auto-calibrated gradient delay estimation from few spokes using radial intersections (RING). *Magn Reson Med*. 2019;81:1898-1906.
49. Hansen MS, Sørensen TS. Gadgetron: an open source framework for medical image reconstruction. *Magn Reson Med*. 2013;69:1768-1776.
50. Xue H, Davies R, Hansen D, et al. Gadgetron inline AI: effective model inference on MR scanner. *Proc Intl Soc Mag Reson Med*. 2019;27:4837.
51. Küstner T, Fuin N, Hammernik K, et al. CINENet: deep learning-based 3D cardiac CINE MRI reconstruction with multi-coil complex-valued 4D spatio-temporal convolutions. *Sci Rep*. 2020;10:1.
52. Terpstra M, Maspero M, Verhoeff J, Berg C. Accelerated respiratory-resolved 4D-MRI with separable spatio-temporal neural networks. *Proc Intl Soc Mag Reson Med*. 2022;30:0305.
53. Diakite M, Campbell-Washburn AE, Xue H. Integration of the BART toolbox into Gadgetron streaming framework for inline cloud-based reconstruction. *Proc Intl Soc Mag Reson Med*. 2018;26:2861.
54. Block KT, Sodickson DK. Yarra: an open software framework for clinical evaluation of reconstruction prototypes. Proceedings of the ISMRM Workshop on Data Sampling & Image Reconstruction; 2016; Sedona.

SUPPORTING INFORMATION

Additional supporting information may be found in the online version of the article at the publisher's website.

Figure S1. Top: Mean PSNR and SSIM evaluated on the 100 slices of the VarNet evaluation dataset after each training epoch. Bottom: Similarly, MSE of magnitude images evaluated on the training dataset (300 slices) and

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

evaluation dataset (100 slices). Both, the BART and the TensorFlow implementation, show a similar convergence behavior. We assume the slight difference of both metrics in the early epochs result from a different initialization of the weights in both implementations.

Figure S2. (A) An nlop-container assigning the nlop F to GPU X. When the container is called, it changes the CUDA context to the GPU X. CUDA events are used to (asynchronously with respect to the CPU) synchronize the new CUDA context with the old one. The input data is copied to a GPU buffer. Now, the container calls F in the new CUDA context such that all data shared with the derivative is allocated on GPU X. Finally, the output is copied to the output array, before the CUDA context is switched back to the original one. The input and output arrays can be located on the CPU or an arbitrary GPU. **B:** Multi-GPU stacking of nlops F1 and F2. The nlops F1 and F2 are assigned to different GPUs and are called in parallel by the corresponding OMP-threads. Both GPU wrappers are synced with the CUDA context active before entering the OMP parallel region. Before leaving the OMP region, the respective CPU threads are synchronized with the CUDA context such that the CUDA context active after leaving the OMP region can assume that all data is written to the output.

How to cite this article: Blumenthal M, Luo G, Schilling M, Holme HCM, Uecker M. Deep, deep learning with BART. *Magn Reson Med*. 2023;89:678-693. doi: 10.1002/mrm.29485

15222594, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/mrm.29485 by Georg-August-Universitaet, Wiley Online Library on [13/07/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License