Received: 29 June 2023 | Revised: 31 July 2023 | Accepted: 23 August 2023

DOI: 10.1002/pamm.202300078

RESEARCH ARTICLE

Model order reduction methods applied to neural network training

M.A. Freitag[1] | **J.M. Nicolaus**[1] | **M. Redmann**[2]

[1]Institute for Mathematics, University of Potsdam, Potsdam, Germany

[2]Institute for Mathematics, Martin Luther University of Halle-Wittenberg, Halle (Saale), Germany

Correspondence
J.M. Nicolaus, Institute for Mathematics, University of Potsdam,
Karl-Liebknecht-Str. 24-25, 14476 Potsdam, Germany.
Email: jan.martin.nicolaus@uni-potsdam.de

Funding information
Deutsche Forschungsgemeinschaft, Grant/Award Number: 318763901 - SFB1294

Abstract

Neural networks have emerged as powerful and versatile tools in the field of deep learning. As the complexity of the task increases, so do size and architectural complexity of the network, causing compression techniques to become a focus of current research. Parameter truncation can provide a significant reduction in memory and computational complexity. Originating from a model order reduction framework, the Discrete Empirical Interpolation Method is applied to the gradient descent training of neural networks and analyze for important parameters. The approach for various state-of-the-art neural networks is compared to established truncation methods. Further metrics like L_2 and Cross-Entropy Loss, as well as accuracy and compression rate are reported.

1 | PRELIMINARIES

In this paper we want to investigate the approximation of the neural network evaluation by a reduced network, where the truncation is obtained by applying the index selection process of the discrete empirical interpolation method (DEIM), known from the field of model order reduction (MOR). DEIM is a hyper reduction technique used to interpolate nonlinear functions, which accelerates repeated evaluations. This paper is structured as follows. Section 1.1 provides an introduction to projection based MOR via the proper orthogonal decomposition (POD) method. Section 1.2 introduces the DEIM approach. Section 2 provides a brief description of neural network truncation, also known as pruning, and describes the application of DEIM in this context. We provide a numerical example and our conclusions in Sections 3 and 4.

1.1 | Projection based model order reduction

Partial differential equations (PDEs) arise in various fields of natural sciences and engineering. Since analytic solutions of general partial differential equations can only be obtained in a small number of special cases, numerical methods for computing an approximation to the exact solution have been developed. Generally, the domain Ω is discretized using methods like finite elements or finite volumes into a finite subset $\{x_1, \dots, x_n\} = \Omega_n \subset \Omega$. For exemplary purposes, let Ω be

the spatial domain on which the dynamics are defined and f the nonlinearity in the PDE

$$\begin{aligned} \partial_t u &= \mathcal{L}u + f(u) \quad \text{in } (0,T)\times\Omega, \\ u(t,x) &= 0 \quad \text{for } (t,x)\in(0,T)\times\partial\Omega, \\ u(0,x) &= u_0(x) \quad \text{for } x\in\Omega, \end{aligned} \tag{1}$$

with Dirichlet boundary conditions. Utilizing, for example, the finite differences discretization scheme, one obtains a system of ordinary differential equations (ODEs)

$$\dot{y} = \mathbf{A}y + F(y), \quad y(0) = y_0 = (u_0(x_1),\dots,u_0(x_n))^T, \tag{2}$$

where $\mathbf{A}\in\mathbb{R}^{n\times n}$ is a constant matrix corresponding to the spatial discretization of the differential operator $\mathcal{L}$ and

$$F : \mathbb{R}^n \to \mathbb{R}^n, y \mapsto F(y) := (f(y_1),\dots,f(y_n))^T, \tag{3}$$

is a (nonlinear) function defined by the componentwise evaluation of f. The dimension of system (2) is determined by the number of discretization points n, which is growing exponentially in the dimension of Ω. In order to numerically solve PDE (1) the system of ODEs (2) is evolved in time. The approximate solution of the original system is then given by $u(t,x_i)\approx y_i(t), i=1,\dots,n$. Clearly, for high spatial resolutions the dimension n of the discretized systems can become very large, which results in expensive computations and high memory complexity.

If in addition a fine time resolution is needed and the evolution equation has to be solved multiple times, for example, for various parameters or initial conditions, computational costs might become unreasonably high. To resolve this issue MOR techniques have been developed, see for example [1–4]. Here, we introduce POD and DEIM.

In the POD framework one seeks to find a set of orthonormal basis vectors $\{v_1,\dots,v_r\}\subset\mathbb{R}^n$, with $r\ll n$, such that the true system state $y(t)$ can be approximated in the "trial subspace" $\mathcal{V}_r = \operatorname{span}\{v_1,\dots,v_r\}$, that is

$$y(t)\approx \mathbf{V}_r y_r(t),$$

for some coefficient vector $y_r(t)\in\mathbb{R}^m$ and $\mathbf{V}_r = [v_1,\dots,v_r]\in\mathbb{R}^{n\times r}$. By substituting the approximation of y into Equation (2) one obtains the approximation

$$\mathbf{V}_r y_r \approx \mathbf{A}\mathbf{V}_r y_r + F(\mathbf{V}_r y_r). \tag{4}$$

To restore equality, the residual $\mathbf{V}_r y_r - \mathbf{A}\mathbf{V}_r y_r - F(\mathbf{V}_r y_r)$ is "tested" against an r dimensional "test subspace" $\mathcal{W}_r\subset\mathbb{R}^n$. In more detail, the Petrov–Galerkin conditions

$$\mathbf{W}_r^T(\mathbf{V}_r y_r - \mathbf{A}\mathbf{V}_r y_r - F(\mathbf{V}_r y_r)) = 0, \quad \mathbf{W}_r = [w_1,\dots,w_r]\in\mathbb{R}^{n\times r}, \tag{5}$$

are enforced, where the subspace $\mathcal{W}_r$ is spanned by the orthonormal columns of $\mathbf{W}_r$. From the Petrov–Galerkin conditions (5) we obtain

$$\mathbf{W}_r^T\mathbf{V}_r y_r = \mathbf{W}_r^T\mathbf{A}\mathbf{V}_r y_r + \mathbf{W}_r^T F(\mathbf{V}_r y_r) \qquad \text{or} \qquad \mathbf{E}_r\dot{y}_r = \mathbf{A}_r y_r + \mathbf{W}_r^T F(\mathbf{V}_r y_r), \tag{6}$$

by redefining $\mathbf{E}_r := \mathbf{W}_r^T\mathbf{V}_r\in\mathbb{R}^{r\times r}$ and $\mathbf{A}_r := \mathbf{W}_r^T\mathbf{A}\mathbf{V}_r\in\mathbb{R}^{r\times r}$. The resulting reduced order model (ROM) (6) is a system consisting of $r\ll n$ variables and equations. If the r is sufficiently small compared to the dimension n of the full order model (FOM), the time required to obtain an approximation of the PDE solution (1) can be reduced significantly. Often it is convenient to choose the *Galerkin projection* $\mathbf{W}_r = \mathbf{V}_r$, which leads to $\mathbf{E}_r = \mathrm{I}_r$.

While it is possible to project the linear system dynamics onto a smaller subspace using POD, the nonlinear part of Equation (6) remains to be evaluated on n components. Since F might not be explicitly known, precomputing $\mathbf{W}_r^T F(\mathbf{V}_r\cdot)$ is likely not possible. The reduction given by POD can therefore generally not accelerate the computation of the costly nonlinear term. Utilising DEIM (see [5]), introduced in the next section, it suffices to compute F on only $k\ll n$ inputs and interpolate the output of the remaining components.

1.2 | The discrete empirical interpolation method

As stated previously, the nonlinear function $\mathbf{W}_r^T F(\mathbf{V}_r \cdot)$ of the projected system dynamics (6) is generally not precomputable. Due to the structure of the nonlinear function F, given by the componentwise evaluation of f, cf. (3), the computational complexity of an evaluation of F is $\mathcal{O}(n\alpha)$, where α is the complexity of a single evaluation of f. Reducing the complexity of F therefore corresponds to either decreasing the number of components f is evaluated at or decreasing α. DEIM [5] finds a set of $k \ll n$ indices $\{i_1, \dots, i_k\} \subset \{1, \dots, n\}$ and constructs a linear interpolation of the form

$$F(y) = \begin{bmatrix} f(y_1) \\ \vdots \\ f(y_n) \end{bmatrix} \approx \mathbf{Q} \begin{bmatrix} f(y_{i_1}) \\ \vdots \\ f(y_{i_k}) \end{bmatrix}.$$

For ease of notation we will denote the simultaneous evaluation of the nonlinear function f on the components index by $\{i_1, \dots, i_k\}$ with $F([y_1, \dots, y_k]^T) = [f(y_{i_1}), \dots, f(y_{i_k})]^T$. Defining $\mathbf{P} \in \mathbb{R}^{n \times k}$ as a row selection matrix, s.t. $\mathbf{P}^T y = [y_{i_1}, \dots, y_{i_k}]^T$, the linear interpolation of $F(y)$ can be written as

$$F(y) \approx \mathbf{Q} F(\mathbf{P}^T y). \tag{7}$$

Here $\mathbf{Q} \in \mathbb{R}^{n \times k}$ is the interpolation matrix corresponding index selection $\{i_1, \dots, i_k\}$. Consequently, the costly function f only has to be evaluated at $k \ll n$ components in each time step. Combined with the complexity of the matrix-vector multiplication of the linear interpolation, one obtains $\mathcal{O}(nk + k\alpha)$ as computational complexity.

In more detail, the DEIM approach approximates the image space $\mathrm{Im}(F) \subset \mathbb{R}^n$ by a linear subspace $\mathcal{U}_k \subset \mathbb{R}^n$ with dimension $\dim(\mathcal{U}_k) = k \ll n$. If $\mathcal{U}_k$ is spanned by orthonormal vectors $\{u_1, \dots, u_k\}$ the evaluation of $F(y)$ can be approximated by a linear combination of these basis vectors

$$F(y) \approx \mathbf{U}_k c(y), \tag{8}$$

where $\mathbf{U}_k = [u_1, \dots, u_k] \in \mathbb{R}^{n \times k}$. The coefficients $c(y) \in \mathbb{R}^k$ can be uniquely determined by selecting k rows in the following way. The index subset $\{i_1, \dots, i_k\}$ is iteratively constructed by selecting the index i_j in the j-th iteration where the interpolation error of the basis vector u_j is maximal. The selection of the interpolation components $\mathbf{P}^T y$ of y in (7) is achieved by collecting standard basis vectors $e_i \in \mathbb{R}^n$ into a matrix $P = [e_{i_1}, \dots, e_{i_k}] \in \mathbb{R}^{n \times k}$, which allows to require exactness in (8), for example

$$\mathbf{P}^T F(y) = \mathbf{P}^T \mathbf{U}_k c(y).$$

Since $\mathbf{P}^T \mathbf{U}_k$ is invertible, the unique solution is given by $c(y) = (\mathbf{P}^T \mathbf{U}_k)^{-1} \mathbf{P}^T F(y)$, resulting in

$$F(y) \approx \mathbf{U}_k (\mathbf{P}^T \mathbf{U}_k)^{-1} \mathbf{P}^T F(y).$$

Defining $\mathbf{Q} = \mathbf{U}_k (\mathbf{P}^T \mathbf{U}_k)^{-1}$ and utilising the componentwise structure of F, approximation (7) is obtained.

The advantage of utilizing DEIM in conjunction with POD is that no additional computations are required, if the subspaces $\mathcal{V}_r$ and $\mathcal{U}_k$ are constructed in the following way. First, singular value decompositions (SVD) of the snapshot matrices

$$\mathbf{X} := \big[y(t_1), \dots, y(t_s)\big] = \mathbb{V}_X \mathbb{S}_X \mathbb{W}_X^T \quad \text{and} \quad \mathbf{F} := \big[F(y(t_1)), \dots, F(y(t_s))\big] = \mathbb{V}_F \mathbb{S}_F \mathbb{W}_F^T,$$

of the system state and the evaluation of F at observation times $t_1, \dots, t_s$ are computed. Let l_1, l_2 be the ranks of $\mathbf{X}$ and $\mathbf{F}$, respectively, then the matrices of left singular vectors are given by the orthonormal columns of the matrices $\mathbb{V}_X = [v_1, \dots, v_{l_1}] \in \mathbb{R}^{n \times l_1}$ and $\mathbb{V}_F = [\hat{v}_1, \dots, \hat{v}_{l_2}] \in \mathbb{R}^{n \times l_l}$. The right singular vectors form the orthonormal columns of the matrices $\mathbb{W}_X \in \mathbb{R}^{n \times l_1}$ and $\mathbb{W}_F \in \mathbb{R}^{n \times l_l}$. The singular values are contained in the diagonal matrices $\mathbb{S}_X = \mathrm{diag}(\sigma_{X,1}, \dots \sigma_{X,l_1})$ and $\mathbb{S}_F = \mathrm{diag}(\sigma_{F,1}, \dots \sigma_{F,l_2})$ in decreasing order. The projection matrices $\mathbf{V}_r$ and $\mathbf{U}_k$ are then constructed by taking the r and k leading singular vectors $\mathbf{V}_r = [v_1, \dots, v_r]$ and $\mathbf{U}_k = [\hat{v}_1, \dots, \hat{v}_k]$. The neglected singular values can be used to bound the error of the interpolation by the DEIM Algorithm.

2 | NEURAL NETWORK PRUNING

Neural networks are a powerful class of models used in a wide range of applications, for example, classification tasks, pattern recognition, simulation or surrogate modelling. Since these models often require a high number of parameters for complex tasks and therefore large amounts of computational power and memory, various approaches have been developed to address the computational challenges. These so called "*Pruning*" methods aim to remove parameters or even entire structures, for example, filters in a convolutional layer, while trying to preserve the overall quality of predictions.

Training a neural network consists of finding sets of weights W and biases b, such that a given Loss function $L(W, b)$, for example, the following L_2 Loss

$$L(W, b) := \sum_{(x,y)\in(\mathcal{X}\times\mathcal{Y})} ||N(x, W, b) - y||_2^2,$$

based on the Euclidean norm $||\cdot||_2$ is minimized, where $N(x, W, b)$ is used to indicate the evaluation of the described neural network architecture with weights W and biases b on the input data $x \in \mathcal{X}$, the target output data is denoted by $y \in \mathcal{Y}$. Originating in the setting of optimization, the gradient descend method serves as the basis for more sophisticated optimization algorithms, like BFGS or the Adam optimizer. Given a set of parameters $p^{(n)} = \text{vec}(W^{(n)}, b^{(n)}) \in \mathbb{R}^{n_p}$ at the n-th gradient descend training step, an update is performed according to

$$p^{(n+1)} = p^{(n)} - \eta \nabla_p L(p^{(n)}), \tag{9}$$

where $\eta > 0$ is called the learning rate. In the context of ODEs, Equation (9) can be regarded as an Euler discretization of the gradient flow

$$\dot{p} = -\nabla_p L(p), \tag{10}$$

with stepsize $\eta > 0$. Generally, the gradient flow (10) does not contain an immediately accessible linear component, causing the POD approximation (6)

$$\dot{p}_r = -\mathbf{W}_r^T \nabla_p L(\mathbf{V}_r p_r).$$

not to be precomputable, which leads us to the use of the hyper reduction technique DEIM, described in Section 1.2. However, since the evaluation of a neural network N is inextricably linked to its structure, performing an intrusive truncation method like POD or DEIM is infeasible. If the objective is to obtain a ROM, that can be treated again as a neural network; an approach is to identify an "important" set of parameters $p_{i_1}, \dots, p_{i_k}$, sparsifying the model accordingly by defining a mask $M \in \{0, 1\}^{n_p}$, s.t.

$$\tilde{p}_i := (p \circ M)_i = \begin{cases} p_i, & i \in \{i_1, \dots, i_k\}, \\ 0, & \text{else} \end{cases}$$

where $\circ$ denotes the Hadamard product, and using $f(x, \tilde{p})$ as an approximation of $f(x, p)$. Such methods are referred to as "pruning" [6]. Current methods prune a network at initialization or after training. Pruning at initialization [7] selects parameters after the network is initialised, but before the actual training occurs. In contrast, for pruning after training [8], a network is trained until convergence in a first step. Next, the parameters p are scored by computing a saliency function $S : \mathbb{R}^{n_p} \to \mathbb{R}^{n_p}$ and truncated, that is, set to zero, if their score is below a selected threshold. Subsequently fine tuning of the network is performed, which means that the network is again trained until convergence to allow recovery. This procedure is repeated until the desired compression is achieved. The authors of [8] summarize the "pruning after training" method by Algorithm 1.

Common baseline methods for benchmarking [8] are magnitude pruning (MP) and gradient magnitude pruning (GMP), with saliency functions componentwise given by

$$S_i^{MP}(p) = |p_i| \quad \text{and} \quad S_i^{GMP}(p) = |p_i \cdot \partial_{p_i} L(p)|. \tag{11}$$

ALGORITHM 1 Pruning after training

$p \leftarrow$ initialize network
$p \leftarrow$ train until convergence criterion is reached
for $i = 1 : \text{NumPruningIterations}$ **do**
  $s(p) \leftarrow$ score p
  $M \leftarrow$ construct mask from $s(p)$
  $p \leftarrow$ fine tune $p \circ M$
end for

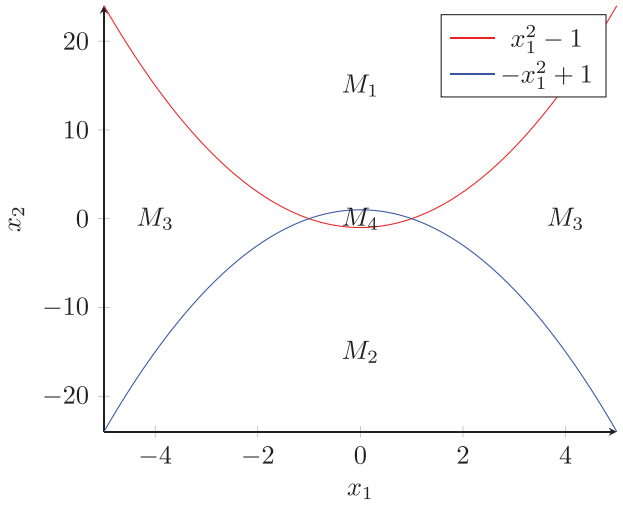


FIGURE 1 Subdivision of $[-4, 4] \times [-20, 20]$ into M_1, M_2, M_3, M_4.

Another reference method is random pruning (RP), in which in each pruning pass a fixed number of randomly chosen weights are removed. We propose to precompute the order in which the weights are pruned by utilization of the DEIM index selection procedure. By defining the nonlinear function F as the gradient of the Loss

$$F(p) = -\nabla_p L(p), \quad F : \mathbb{R}^{n_p} \to \mathbb{R}^{n_p}, \tag{12}$$

we can obtain basis functions of the image of F by training the neural network for s steps and collecting the corresponding states into a snapshot matrix $\mathbf{F}$ and its singular value decomposition

$$\mathbf{F} = \left[F(p^{(1)}), \dots, F(p^{(s)})\right] = \left[-\nabla_p L(p^{(1)}), \dots, -\nabla_p L(p^{(s)})\right] = \mathbf{USV}^T. \tag{13}$$

In contrast to Section 1.2, a non-truncated SVD of $\mathbf{F}$ is performed to obtain a reordering $\{p_{i_1}, \dots, p_{i_n}\}$ of all parameter components $\{p_1, \dots, p_{n_p}\}$. Subsequently, the index selection process of DEIM is performed to obtain $\{i_1, \dots, i_n\}$.

3 | COMPARISON TO BASELINE METHODS

In our numerical experiment, a fully connect feedforward neural network with 4 layers, containing 20 neurons each with logistic sigmoid activation functions and four output neurons, is investigated. The total number of network parameters is hence $n_p = 984$. The network was trained on the simple classification task to assign the label $e_i \in \mathbb{R}^4$ to the point (x_1, x_2) if $(x_1, x_2) \in M_i$, cf. Figure 1. The network was trained using gradient descent over $s = 10^4$ epochs, a training set consisting of 100 randomly chosen datapoints and the standard L_2 Loss. To construct the snapshot matrix $\mathbf{F}$ from Equation (13), all gradients $F(p^{(i)}) = -\nabla_p L(p^{(i)}))$, $i = 1, \dots, s$ were collected during the training. To allow a better exploration of the image space of F, $n_a s_a$ additional gradients from n_a short training trajectories, each consisting of s_a gradient descend steps, were included.

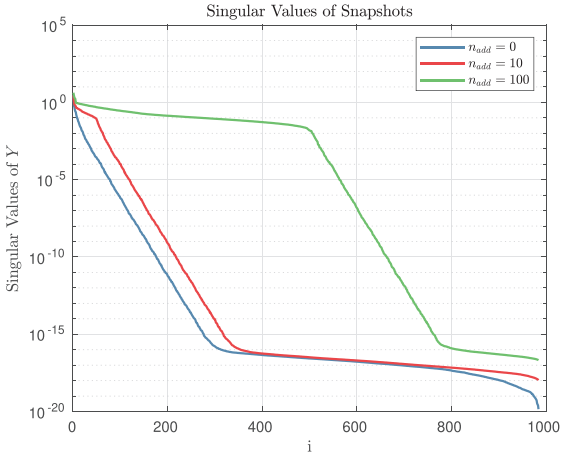
ALGORITHM 2 Construction of nonlinear snapshot matric $\mathbf{F}$

$p \leftarrow$ initialize network with random weights
$\mathbf{F} \leftarrow [\quad]$
for $i = 1 : s$ **do**
  compute $F(p) = -\nabla_p L(p)$
  $\mathbf{F} \leftarrow [\mathbf{F}, F(p)]$
  $p \leftarrow p + F(p)$
end for
for $i = 1 : n_a$ **do**
  $p \leftarrow$ initialize network with random weights
  for $j = 1 : s_a$ **do**
    compute $F(p) = -\nabla_p L(p)$
    $\mathbf{F} \leftarrow [\mathbf{F}, F(p)]$
    $p \leftarrow p + F(p)$
  end for
end for

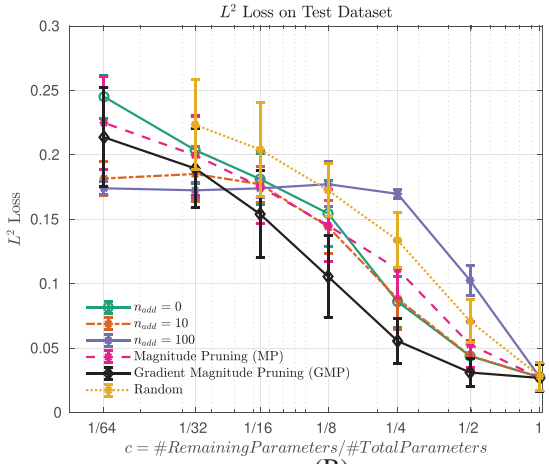
ALGORITHM 3 Pruning experiment

for $j = 1 : K$ **do**
  $p \leftarrow$ initialize network with random weights
  $p \leftarrow$ train network until convergence criterion is reached
  for method in PruningMethods **do**
    $\hat{p} \leftarrow p$
    for $i = 1 :$ NumPruningIterations **do**
      $s(\hat{p}) \leftarrow$ score $\hat{p}$
      $M \leftarrow$ construct mask from $s(\hat{p})$
      $\hat{p} \leftarrow$ fine tune $p \circ M$
      compute metrics for specified compression rates
    end for
  end for
end for
compute empiric mean and empiric standard deviation for each metric and compression rate

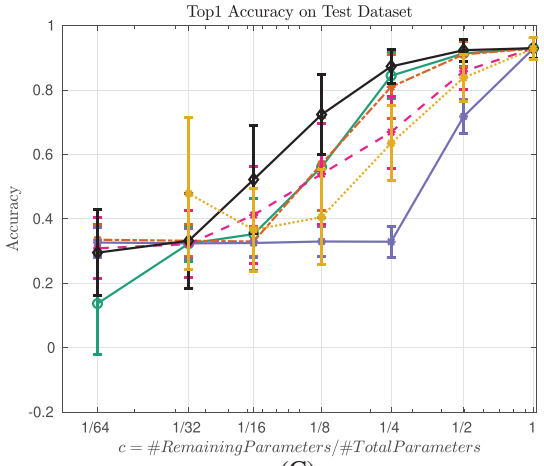
The snapshot matrix $\mathbf{F}$, as constructed by Algorithm 2, therefore consists of n_p rows and $s + n_a s_a$ columns. We compare our approach for $n_a \in \{0, 10, 100\}$ and $s_a = 10$ to the baseline methods mentioned in the previous section, cf. Equation (11). The experiment is set up as follows. First, Algorithm 2 is performed with $n_a = 100$, returning the nonlinear snapshot matrix $\mathbf{F}_{100}$. From $\mathbf{F}_{100}$, the submatrices $\mathbf{F}_0$ and $\mathbf{F}_{10}$, consisting of the first s and $s + 10s_a$ columns are extracted. These submatrices correspond to the choice $n_a = 0$ and $n_a = 10$, respectively. For each matrix $\mathbf{F}_0, \mathbf{F}_{10}$ and $\mathbf{F}_{100}$, the first n_p left singular vectors are computed. Subsequently the index selection process of the DEIM Algorithm is performed on each set of singular vectors, cf. Equation (13) at the end of the previous section and [5]. Afterwards Algorithm 3, an extension of Algorithm 1, is performed. After the training of the randomly initialized network has converged, each pruning method is performed according to the second part of Algorithm 1. For each pruning method the L_2 Loss, Top1 accuracy, that is, the ratio of correct predictions to total number of predictions, are reported. In addition we report the Cross-Entropy Loss $-\sum_{i=1}^{100} y_i^T \log(q_i)$, where $\log(q_i)$ is the componentwise natural logarithm of q_i, which is the softmax rescaled output $N(x_i, W, b)$ and y_i is the label corresponding to x_i. The metrics are computed at the percentages of remaining network parameters $c \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$. The experiment is repeated $K = 128$ times. To obtain the plots in Figure 2, the sample mean and sample standard deviation are computed for each pruning method and each compression rate.



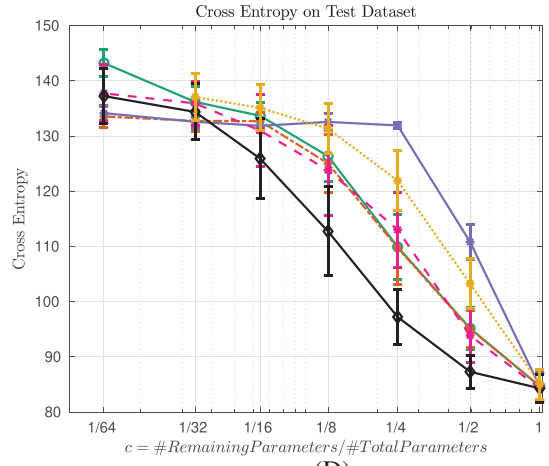
(A)
Singular values of nonlinear snapshot matrices



(B)
Sample mean and standard deviation of L_2 Loss



(C)
Sample mean and standard deviation of Top1 accuracy metric



(D)
Sample mean and standard deviation of Cross-Entropy metric

FIGURE 2 L_2 Loss, Top1 accuracy and Cross-Entropy metrics as computed by Algorithm 3 for methods GMP,MP, RP as well as our method for $n_a = 0, 10, 100$ and singular values of snapshot matrices $\mathbf{F}_0, \mathbf{F}_{10}, \mathbf{F}_{100}$.

4 | CONCLUSIONS AND FURTHER RESEARCH

We would like to point out, that the index ordering $\{i_1, \dots, i_n\}$ is fixed before Algorithm 3 is performed and does not depend on the weights of the neural network at the time of pruning. Therefore, in contrast to the pruning methods used for comparison, our approach does not adapt to the state of the neural network during the pruning procedure.

An immediate observation in Figure 2(A) is that the decay of singular values of $\mathbf{F}_0$ is shifted by a slower initial decay, when additional trajectories ($n_a = 10, 100$) are incorporated into the snapshot matrix. In the case, where no additional trajectories were incorporated ($n_a = 0$), our approach outperforms MP for low compression rates $c = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$ and is comparable to GMP for $c = 1, \frac{1}{2}, \frac{1}{4}$ in the Top1 accuracy metric with overall similar standard deviations compared to the reference methods, except random pruning. The reference method GMP outperforms all other methods for $c \geq \frac{1}{16}$. Our approach decreases over all metrics for $c \geq \frac{1}{16}$, when using additional trajectories in the snapshot matrices ($n_a = 10, 100$). In the case of high compression rates $c = \frac{1}{32}, \frac{1}{64}$, the pruning order derived from $\mathbf{F}_{10}$ and $\mathbf{F}_{100}$ show better scores over all metrices. Furthermore, we observe a reduced standard deviation for our approach compared to the reference methods GMP and MP by up to one order of magnitude.

The results of the reported experiments, combined with the non-adaptivity of our approach to the network state during the pruning stage, suggest that an identification of an approximating sub network from the networks gradients is possible.

Since we only conducted our experiments for a neural network of simple structure, further research, investigating the proposed approach for more complex structures, is needed. The decrease of standard deviation and overall comparable or better performance for $n_a = 100$ and $c = \frac{1}{64}, \frac{1}{32}$ suggest the existence and identification of an "essential" or "worst-case" subnetwork. A possible connection to the pathway decomposition of the neural tangent kernel, proposed in [7] is of interest. In addition, the view of residual neural networks in the context of dynamical systems [9] might allow an application of other MOR techniques.

ACKNOWLEDGMENTS

The research has been partially funded by the Deutsche Forschungsgemeinschaft (DFG) - Project-ID 318763901 - SFB1294. In particular, we would like to acknowledge the productive discussions at the annual SFB1294 Spring School 2023. Furthermore, we thank Dr. Thomas Mach for insightful discussions.

Open access funding enabled and organized by Projekt DEAL.

CONFLICT OF INTEREST STATEMENT

The authors declare that they have no conflict of interest.

DATA AVAILABILITY STATEMENT

The code is available at https://github.com/JMNicolaus/InterpolationPruning.

ORCID

J.M. Nicolaus https://orcid.org/0009-0005-1325-3626

REFERENCES

1. Antoulas, A. C., Beattie, C. A., & Güğercin, S. (2020). Interpolatory methods for model reduction. In D. Estep (Ed.), *Computational science & engineering*. Society for Industrial and Applied Mathematics (SIAM).
2. Antoulas, A. C., Gosea, I. V., & Ionita, A. C. (2016). Model reduction of bilinear systems in the Loewner framework. *SIAM Journal of Scientific Computing*, *38*(5), B889–B916.
3. Benner, P., Gugercin, S., & Willcox, K. (2015). A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review*, *57*(4), 483–531.
4. Benner, P., Cohen, A., Ohlberger, M., & Willcox, K. (Eds.). (2017). Model reduction and approximation. In *Computational science & engineering* (Vol. 15). Society for Industrial and Applied Mathematics (SIAM).
5. Chaturantabut, S., & Sorensen, D. C. (2010). Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal of Scientific Computing*, *32*(5), 2737–2764.
6. Reed, R. (1993). Pruning algorithms-A survey. *IEEE Transactions on Neural Networks*, *4*(5), 740–747.
7. Gebhart, T., Saxena, U., & Schrater, P. (2021). A unified paths perspective for pruning at initialization. *arXiv:2101.10552*.
8. Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., & Guttag, J. (2020). What is the state of neural network pruning? *Proceedings of Machine Learning and Systems*, *2*, 129–146.
9. Berner, J., Grohs, P., Kutyniok, G., & Petersen, P. (2021). The modern mathematics of deep learning. In P. Grohs & G. Kutyniok (Eds.), *Mathematical aspects of deep learning*. Cambridge University Press.

How to cite this article: Freitag, M. A., Nicolaus, J. M., & Redmann, M. (2023). Model order reduction methods applied to neural network training. *Proceedings in Applied Mathematics and Mechanics*, *23,* e202300078. https://doi.org/10.1002/pamm.202300078