Neural Networks with Sequentially Semiseparable Weight Matrices

Matthias Kissel

Matthias Kissel. *Neural Networks with Sequentially Semiseparable Weight Matrices.*
Dissertation, Technical University of Munich, Munich, Germany, 2024.

Chair of Data Processing, Technical University of Munich, 80333 Munich, Germany,
https://www.ce.cit.tum.de/ldv/.

Technische Universität München
TUM School of Computation, Information and Technology

Neural Networks with Sequentially Semiseparable Weight Matrices

Matthias Peter Kissel

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Vasilis Ntziachristos

Prüfende der Dissertation:

1. Prof. Dr.-Ing. Klaus Diepold
2. Prof. Dr. Alle-Jan van der Veen

Die Dissertation wurde am 04.09.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 27.03.2024 angenommen.

Abstract

Neural Networks (NNs) are nowadays used in many application areas for solving increasingly complex tasks. Therefore, more and more resources are required for training and deployment. One approach to reduce the resource requirements is to use structured matrices in NNs. In this thesis, I study NNs whose weight matrices have a particular structure, namely they are sequentially semiseparable (SSS). I show that for NNs with SSS weight matrices, computational resources required for inference can be reduced. In addition, such networks can achieve better prediction accuracy compared to their standard counterparts depending on the problem at hand. The performance depends on the method used to bring the structure into the network, for which I compare three different approaches. Lastly, I investigate how the behavior compares to NNs with structured weight matrices of different types. The experiments show that the achieved results depend on the chosen structure. Accordingly, the number of parameters need not be the dominant criterion for prediction accuracy. The choice of a suitable structure for a given task also plays an important role.

Kurzzusammenfassung

Neuronale Netze werden heute in vielen Anwendungsbereichen zur Lösung immer komplexerer Aufgaben eingesetzt. Hierbei werden für das Training und den Einsatz der Netze immer mehr Rechenressourcen benötigt. Ein Ansatz zur Reduzierung dieses Ressourcenbedarfs ist die Verwendung strukturierter Matrizen. In dieser Arbeit untersuche ich den Einsatz von Matrizen mit einer bestimmten Struktur in neuronalen Netzen: Sequentiell Semiseparable (SSS) Matrizen. Ich zeige, dass für neuronale Netze mit SSS Gewichtsmatrizen die für die Inferenz erforderlichen Rechenressourcen reduziert werden können. Darüber hinaus können solche Netze je nach Problemstellung eine bessere Vorhersagegenauigkeit als Netze mit unstrukturierten Gewichtsmatrizen erreichen. Die Genauigkeit hängt von der Methode ab, mit der die Struktur in das Netz eingebracht wird. Im Vergleich mit neuronalen Netzen mit strukturierten Gewichtsmatrizen anderer Strukturklassen stellt sich heraus, dass die erzielten Ergebnisse von der gewählten Struktur abhängen. Demnach muss die Anzahl der Parameter nicht das ausschlaggebende Kriterium für die Vorhersagegenauigkeit sein. Auch die Wahl einer geeigneten Struktur passend zur gegebenen Aufgabenstellung spielt eine wichtige Rolle.

Acknowledgements

I am very grateful that Prof. Klaus Diepold offered me to join the Chair of Data Processing and write my thesis under his guidance. Prof. Diepold inspired me for the topic and the discussions with him have always been of great help. Thanks also to all my colleagues at the chair for their valuable feedback and support.

At this point I would also like to thank my parents, Beate and Peter Kissel, who have always supported and been there for me. They taught me to think critically, which has set the foundation for my interest in science.

Finally, I would like to thank my wife, Alexandra Kissel, for her support over the years.

Contents

1. Motivation

1.1. Trade-off between Complexity and Efficiency

In the domain of machine learning, there is often a fine line between necessary complexity and efficient use of resources. However, it is especially important in today's world to set this trade-off appropriately. This is due to the fact that besides the savings through reduced server costs for training or deploying a model, we are also interested in the impact on the environment. The goal is to eliminate unnecessary computational effort, for example in order to reduce the CO_2 fingerprint of a model.

The research of this thesis focuses on this line between complexity and efficiency. My focus is on saving computational operations when using modern NNs. These networks are often particularly large and computationally expensive to train and to use. Here, we are interested in investigating not only possible savings but also the impact on the prediction accuracy of the network.

There are several approaches to make the training and deployment of NNs more computationally efficient. My approach to achieve these computational savings is to exploit structures in the networks. Specifically, I am looking at matrix structures that allow computations for matrices to be performed with fewer operations and to store the matrix with fewer number of parameters (taking advantage of the structure). In the following, I first give a motivational example for a case, where resources can be spared by using structured matrices in NNs. Subsequently, I give an overview over the computational problems for modern NNs. These problems can be tackled by using structured matrices, which are introduced in the following section. Finally, I present the outline of this thesis.

1.2. Motivational Example: Neural Networks controlling Drones

In this work, I investigate methods that can make the use of NNs more efficient. To illustrate what this means, I use a running example at some passages. It is about an autonomously flying quadrotor drone, which is controlled by an NN. This setting serves as an application example for the use of NNs on resource constrained hardware. In order to robustly control a drone, the sensory input data must be quickly evaluated and appropriate motor outputs must be set in real time. The drone considered in my

Takeoff weight	28.6g
Onboard Microcontroller	STM32F405 (168 MHz, 192kb SRAM, 1Mb flash)
Onboard Sensors	3 axis accelerometer / gyroscope, z-axis LIDAR sensor
Flight time with stock battery	7 minutes

Table 1.1.: Specifications of the Crazyflie 2.1 drone.

example is the Crazyflie 2.1[1] drone, which is often used in research projects. The drone has an onboard microcontroller with 168MHz processing power and 1Mb flash memory (more specifications are given in Table 1.1).

In my example, I refer to the experimental setup described in [35]. In this setting, a standard fully-connected Feed-Forward Neural Network (FFNN) is used to compute the pulse width modulation (PWM) signals for the quadrotor motors. For this, sensor inputs from the gyroscope as well as the accelerometer are used as inputs to the NN (together with additional information like the deviation from the target position and the previous action taken by the NN). In order to robustly control the drone, control inputs must be delivered at high frequency (typically around 1000Hz). Therefore, the duration for the NN inference should be in the order of 1ms, whereas all computations are performed on the microcontroller of the drone. Besides the computation resources, the energy resources are also restricted. Using the onboard battery, the drone can fly for about 7 minutes without recharging in the standard mode (which means without NN controller). This flight time might be reduced, if more energy is consumed for performing NN inference on the microcontroller of the drone. There are two possible design criteria for the NN controller:

- Reducing the inference time in order to increase the frequency of the control signals
- Reducing the ressource consumption per evaluation (and hence the energy consumption) at fixed frequency of the control signals

The choice of the design criterion depends on the target application.

Potential real-world applications for NN controlled drones are search and rescue missions [61, 89], detecting and tracking animals [9, 66], or spotting wild fires [44, 58]. In this thesis, I do not focus on a specific application domain. Therefore, I also do not address concerns about regulations or safety of autonomous flying drones. When comparing the resource consumption of the standard approach with methods proposed in this thesis, I assume that the power consumption and inference time is proportional to the number of operations performed. In a real world setting, this might not exactly

[1] `https://www.bitcraze.io/products/crazyflie-2-1/`

be the case, since these metrics depend on the implementation of the algorithm and the hardware used.

1.3. Modern Neural Network Architectures

In the last years, NNs have been used to solve increasingly complex tasks. These include for example beating the best human player in the game of Go [71], generating images, text or music [6, 8, 46], or achieving remarkable results in the domain of image classification [41, 53, 70, 76]. However, as the difficulty of the tasks increases, so does the complexity of the networks required to accomplish them. Therefore, modern NN architectures often comprise millions (or even billions) of parameters.

The increased number of parameters in the network comes with multiple challenges [50], including the following.

- The training and inference times increase with the number of parameters, since more operations have to be performed in order to propagate information through the network. Therefore, modern NNs are often trained and deployed to specialized hardware, where the training phase can still last for several weeks [71]. With respect to the example of an NN controlled drone, this means that the time between sensor readouts and the motor outputs increases with larger NNs. For example, we measured in our paper [52] that inference with a network containing layers with 30 hidden neurons takes 0.4ms on the drone microcontroller. In comparison, inference with a network comprising 6 hidden neurons takes only 0.06ms.

- As the number of operations required for inference increases, so does the amount of energy consumed. In the drone example, the power consumption directly affects the flight time, since the onboard battery has a limited capacity. In our experiments, it has been shown that the flight time can vary up to 20 seconds (approximately 5% of the standard flight time) depending on how the drone is controlled. The energy consumption can also be an issue for applications, which are not battery driven. This is due to the fact that higher energy costs lead to higher operating costs. Moreover, CO_2 emissions can increase with energy consumption, thus contribute to today's climate change [74].

- The memory requirements of modern NNs can be an issue depending on the available hardware setting. For example, the MobileNet V2 model designed for computer vision tasks on mobile devices has 3.4 million parameters, requiring more than 12MB in the Imagenet pretrained version provided by Google[2]. In comparison, there is only 1Mb of flash memory and 100kb of SRAM available

[2] https://tfhub.dev/google/imagenet/mobilenet_v2_100_224/classification/5

on the microcontroller of the drone example. This means that such a large Convolutional Neural Network (CNN) does not fit into the working memory of the microcontroller, nor into the available internal memory.

- In addition to storage requirements, bandwidth requirements related to storage operations can also be a bottleneck [75]. Regarding the drone example, the time needed for copying data from flash memory to SRAM can play a role for the NN inference time. This is especially the case if the whole NN does not fit into the working memory. Even if inference takes place on a GPU, memory bandwidth can be a bottleneck, whereas on-chip memory bandwidth plays the major role [42].

The presented challenges are particularly severe for applications targeting mobile devices or embedded hardware. In this case, the available computational and memory resources are limited. Despite these challenges, the demand for applications using NNs on resource constrained hardware is increasing. This includes, for example, face detection algorithms on smartphones [82] or edge computing applications [13].

Matrices play a major role in the resource requirements of NNs. In densely connected FFNNs, the parameters of each network layer are grouped in matrices (referred to as weight matrices in the following). For storing such a weight matrix $W \in \mathbb{R}^{n \times m}$, nm values have to be specified. In addition, performing inference with the network requires computational operations for multiplying the weight matrix of a layer with its inputs, which are grouped into vectors. These matrix-vector multiplications require $O(nm)$ operations in general.

1.4. Structured Matrices

Not all matrices $W \in \mathbb{R}^{n \times m}$ require nm parameters to be stored in order to be fully defined. This is the case for *data sparse* matrices. The entries of data sparse matrices have a certain relationship to each other, which we denote as structure.

Definition 1 *A matrix $A \in \mathbb{R}^{m \times n}$ is called data sparse or structured, if it is defined by less than $O(mn)$ parameters.*

Definition 1 allows different orders of magnitude for the number of parameters required for defining the structured matrix. For example, for the case $n = m$, there are many structure types which require $O(n)$ parameters. These structures often show their advantages already for small matrix dimension n. In contrast, other structure types require $O(n \log^{\kappa} n)$ parameters (for $\kappa \in \mathbb{N}$). Algorithms for such structures often require large matrix dimensions n in order to being more efficient than the standard algorithms for unstructured matrices.

Note that *data sparse* matrices need not be *sparse*. Sparse matrices, in contrast to *data sparse* matrices, contain many zero-valued entries.

Definition 2 *A matrix $A \in \mathbb{R}^{m \times n}$ is called sparse, if it has less than $O(nm)$ nonzero entries.*

Both, *sparse* and *data sparse* matrices, can be described with less than $O(mn)$ parameters. However, in structured matrices, there is no need that any entry is zero. Instead, due to the structure in the matrix, fewer parameters are required in order to define the whole matrix.

For some matrix structures, not only parameters can be saved, but also efficient algorithms exist for multiplying the matrix with a vector. This results in a subquadratic order of operations required for computing the matrix-vector product as well as parameters required for storing the matrix.

The advantages of structured matrices can also be used in NNs if the weight matrices are structured. In this case, fewer parameters are needed to store the weight matrices. Moreover, computational resources can be saved when information is propagated through the network. Operations can be saved due to the reduced effort required to compute the product between the structured weight matrix and the input vector. This can lead to smaller inference times as well as energy savings.

There are many different types of structures that can be present in a matrix. We introduce the four main structure classes in our survey paper [50]: SSS matrices, hierarchical matrices, matrices of low displacement rank, and products of sparse matrices. These classes are based on different types of structures, which means that the relationship between the elements in the matrix differs. In this thesis, I focus on SSS matrices, which originate from time-varying systems theory [21]. This structure is defined in Section 3.1.

1.5. Goal and Thesis Outline

In this thesis, I set the focus on SSS matrices applied to NNs. I am interested in how NNs behave when their weight matrices are sequentially semiseparable. This refers to the prediction accuracy as well as the potential savings in the required memory and computational resources. Moreover, I investigate ways to bring the structure into the fully-connected layers of NNs and compare the observed effects of using SSS weight matrices to using other types of structured matrices in NNs.

The remainder of this thesis is organized as follows. I first give an overview over existing work about structured matrices and how they are used in NNs in Chapter 2. Then, I introduce the concepts on which this thesis is built in Chapter 3. This includes the class of SSS matrices, and different architectures of NNs and how they are trained traditionally. In the subsequent Chapter 4, I introduce my research questions and hypotheses. The methods used for my experiments are introduced in Chapter 6. Here, I show how SSS matrices can be approximated starting from trained weight matrices, or how they can be trained using gradient descent. In the following Chapter 7, I discuss the answers to my research questions and determine for each hypothesis whether it can be falsified or verified. This is based on observations and results obtained from our previous publications. Finally, I conclude my findings in Chapter 8. The summaries of my first author publications, on which this thesis is based, can be found in the appendix.

2. Related Work

2.1. Matrices of Low Displacement Rank

Arguably the best known structure class are matrices of low displacement rank [63]. This structure class includes Toeplitz, Hankel, Vandermonde and Cauchy matrices, which arise in many linear algebra problems. The underlying idea is that the entries of a matrix M might be shifted and modified versions of other entries in the matrix [63]. This means, that after applying the *operator matrices* A and B, the resulting matrix $L(M)$ has low rank. Depending on the type of the displacement, *applying* the operator matrices is defined differently. For displacement operators of the Sylvester type, $L(M)$ is given by

$$L(M) = \nabla_{A,B}(M) = AM - MB. \tag{2.1}$$

Correspondingly, $L(M)$ is given by

$$L(M) = \Delta_{A,B}(M) = M - AMB \tag{2.2}$$

for displacement operators of the Stein type.

Matrices of low displacement rank occur and have been used in many linear algebra applications [63]. This includes algorithms for adaptive filtering [45], for solving systems of equations [7, 36], and algebraic computations in general [3, 64]. For example, Gustavson and Yun [36] presented an algorithm, which can be used to solve a Toeplitz system of linear equation with $O(n \log^2 n)$ operations. Pan [63] gives a comprehensive overview over applications and algorithms using or based on matrices of low displacement rank.

One of the most popular network architectures for NNs, the CNN, is implicitly based on matrices of low displacement rank. This is, because the weight matrices of CNNs can be represented as sparse Toeplitz Matrices. In the domain of images, this architecture has been demonstrated to yield better performance in terms of computational requirements as well as generalization capabilities compared to traditional NNs [55, 76]. Other works focus on explicitly using weight matrices of low displacement rank in NNs. For example, Sindhwani et al. [73] used Toeplitz-like weight matrices in NNs. In their end-to-end training pipeline, they optimized the low rank matrices of the displacement representation, while fixing the operator matrices. In contrast, Thomas et al. [78] introduced a class of matrices of low displacement rank, which facilitates training the operator matrices end-to-end together with the low-rank components. In addition to these results concerning the practical use and training of neural with weight matrices of low displacement rank, there exist also theoretical results. For example, Zhao et al. [87] analyzed the properties of these networks, showing for example that the universal approximation theorem holds for NNs with weight matrices of low displacement rank. Another proof is given by Liu et al. [60]. They showed that the universal approximation theorem holds for Toeplitz or Hankel weight matrices in NNs with arbitrary width and fixed depth, as well as in NNs with arbitrary depth and fixed width.

2.2. Hierarchical Matrices

The idea behind Hierarchical matrices ($\mathcal{H}$ matrices) [37] is that even if the overall matrix has full rank, there might still be parts in the matrix which have a low rank. If these low rank parts are taken into account when storing the matrix and performing computations, storage and computational resources can be saved. In order to represent a matrix as $\mathcal{H}$-matrix, the parts of the matrix are arranged in a tree (the so called block cluster tree), so that the matrices at the leaves of the tree are either small or have a low rank [37]. In this representation, the matrix can be efficiently multiplied with a vector. For this purpose, the individual leaves of the block cluster tree are multiplied with the corresponding entries in the vector (making use of the low-rank property of the leaves). Subsequently, the intermediate results are merged to obtain the overall result of the matrix-vector multiplication.

There are several application domains in which $\mathcal{H}$-matrices are used [37, 38]. This includes, for example the efficient treatment of discrete integral equations [5], support solving eigenvalue problems [27], finite element methods [88], and solving large scale algebraic matrix Riccati equations [33].

$\mathcal{H}$-matrices have also been used in the domain of NNs. They can, for example, be used as weight matrices in NNs, which results in a multiscale structure in the network [24]. The number of parameters needed to define the weight matrix can be further reduced by using $\mathcal{H}^2$ matrices [23], which are a special type of $\mathcal{H}$-matrices. Besides being used as weight matrices in NNs, $\mathcal{H}$-matrices can be used to speed up the training of the network. For example, Chen et al. [12] proposed to use $\mathcal{H}$-matrices for approximating the Generalized Gauss-Newton Hessian, which can be used to spare resources during (second-order) training, analyzing NNs, and estimating learning rates [12]. Also Ithapu proposed to analyze NNs using $\mathcal{H}$-matrices, by investigating the inter-class relationships of deep learning features using a multi-resolution matrix factorization [43]. Moreover, Wu et al. [84] proposed to compress NNs by applying the Hierarchical Tucker decomposition [32] to NNs.

2.3. Products of Sparse Matrices

Another structure class is given by products of sparse matrices. Note that the product of sparse matrices is not sparse in general. Therefore, many dense matrices can be (approximately) represented by products of sparse matrices [16], which in turn can be beneficial for storing the matrix or performing computations with it. In order to fully define a sparse matrix as defined in Definition 2, the values of the non-zero elements of the matrix must be determined together with their positions in the matrix. These values defining the sparse matrix can be stored in different ways. For example, the compressed sparse column matrix format can be used, which shows good performance for computations performed on CPUs [34].

Sparse matrices arise in various application areas and disciplines [22]. This includes, for example, economic modelling, navier-stokes problems, power network modelling, or astrophysics. Such sparse linear system problems can be solved directly by using iterative methods [69]. Also, products of sparse matrices play a role in linear algebra. For example, both the operators of the widely used Discrete Wavelet Transform as well as the Fourier Transform can be expressed based on products of sparse matrices [1, 56].

Sparse matrices have been used early in NNs [40, 57]. There are different methods for obtaining sparse weight matrices, including regularization [83], pruning techniques [4], and hand-tuned heuristics [19]. In contrast, the research field concerning the use of *products of sparse matrices* in NNs is rather young. In this context, it has been proposed several times to use Butterfly matrices as weight matrices in NNs [1, 15, 16, 59]. Butterfly matrices have a fixed sparsity pattern. Using them in NNs has the advantage that the positions of the non-zero elements are fixed throughout the training. This makes NNs comprising Butterfly weight matrices trainable end-to-end, by avoiding the non-differentiable problem of finding the right sparsity pattern. Instead of training the NN with products of sparse matrices end-to-end, the products of sparse matrices can also be identified by approximation. For this purpose, Magoarou et al. [56] presented an algorithm that can be used to approximate matrices with products of sparse matrices. Based on this algorithm, Giffon et al. [28] showed that CNNs with products of sparse matrices as weight matrices can yield better accuracy-compression trade-offs than other popular NN compression methods.

2.4. Sequentially Semiseparable Matrices

First publications touching the concepts of semiseparable matrices date back to 1937 [25, 81]. SSS matrices, in particular, are relevant for various application domains like computational science or engineering. For example, SSS matrices occur when describing time-varying systems using a state-space representation [21]. They have a block structure defined by multiple smaller matrices, whereas the block matrices describe the input-output behavior of a time-varying system at different time steps. This is explained in more detail in Section 3.1, where I formally define SSS matrices.

SSS matrices have some interesting properties. Furthermore, there are efficient algorithms for performing various linear algebra operations with such matrices [81]. This

	Structured Weight Matrices	Other Approaches
Matrices of Low Displacement Rank	Sindhwani et al. [73] Thomas et al. [78] Zhao et al. [87] Liu et al. [60]	-
$\mathcal{H}$-matrices	Fan et al. [24] Fan et al. [23] Wu et al. [84]	Ithapu [43] Chen et al. [12]
Products of Sparse Matrices	Dao et al. [15] Dao et al. [16] Li et al. [59] Ailon et al. [1]	Magoarou et al. [56]
SSS matrices	-	Zamarreno and Vega [86] Van Lint et al. [80] Titti et al. [79]

Table 2.1.: Examples of prior work in which the different structure classes have been used in the domain of NNs.

includes, for example, solving SSS systems of equations [11], or inverting SSS matrices [20]. An important result for SSS matrices is that they can be efficiently multiplied with a vector [10, 30]. Being a special member of the class of semiseperable matrices, other properties which have been shown for general semiseparable matrices also apply to SSS matrices. I refer to [81] for an overview of the results for the general class on semiseparable matrices.

Prior to the work presented in this thesis, SSS matrices have not been used as weight matrices in NNs. Instead, related work using SSS matrices in the context of NNs focused on finding suitable architectures for time-varying system applications. This includes State-Space NNs [80, 86], which introduce non-linearity into the representation of time-varying systems, and Time-Varying NNs [79], whose weights can change over time.

2.5. Summary: Structured Weight Matrices

In the previous four sections, I presented the four main structure classes, which have been used in the context of NNs. I only mentioned the results, which are most important for the scope of this thesis. For a more detailed overview, please refer to our survey paper [50], where we analyzed the structure classes in detail and compared them to each other in two benchmarks.

The contributions for each structure class in the domain investigated in this thesis are summarized in Table 2.1. It is evident, that all structure classes can potentially be used as weight matrices in NNs. However, prior to the work presented in this paper, only three of the four structure classes have been investigated with respect to the effects of using them as weight matrices in NNs.

So far, SSS matrices have been used in classical linear algebra problems. The aim of this work is to also investigate possible applications of SSS matrices in NNs. I chose this structure class for my investigations for two reasons. On the one hand, there is a large body of theory available for SSS matrices, which can be used for theoretical considerations about their use-cases. On the other hand, this class is particularly exciting, since there have been no studies on the possible use of SSS matrices in NNs yet.

3. Background

3.1. Sequentially Semiseparable Matrices

Definition

I introduce SSS matrices in the context of time varying systems. This is intuitive, because when describing time varying systems using state-space methods, the SSS structure naturally occurs. For time-varying systems, the output a_k at the k^{th} time step with $k = 1, \ldots, p$ can be computed by

$$x_{k+1} = A_k x_k + B_k u_k, \tag{3.1}$$

$$\hat{x}_k = E_k \hat{x}_{k+1} + F_k u_{k+1}, \tag{3.2}$$

$$a_k^{(1)} = C_k x_k + D_k u_k, \tag{3.3}$$

$$a_k^{(2)} = G_k \hat{x}_k, \tag{3.4}$$

yielding

$$a_k = a_k^{(1)} + a_k^{(2)}. \tag{3.5}$$

Here, u_k is the input to the system at time step k. x_k and $\hat{x_k}$ are the causal and anti-causal state of the system respectively. The matrices A_k, B_k, C_k, D_k, E_k, F_k and G_k describe the behavior of the system. For example, B_k maps inputs to future states. Note that the dimension of the matrices are not constant. This is due to the fact that state, input, or output dimensions might change over time. In the following, I refer to the k^{th} time step as the k^{th} computation stage, since the matrices considered in this thesis need not to be connected to physical properties.

By concatenating the inputs u_k and outputs a_k into vectors u and a, the input-output behavior of the system can be expressed in operator space

$$a = Tu, \tag{3.6}$$

where T is the SSS operator matrix defined as

$$\begin{aligned} T = D + C(I - ZA)^{-1}ZB \\ + G(I - Z^T E)^{-1} Z^T F. \end{aligned} \tag{3.7}$$

Here, I is the identity matrix and Z is a down-shift matrix

$$Z = \begin{pmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & 0 \\ 0 & & 1 & 0 \end{pmatrix}. \tag{3.8}$$

A, B, C, D, E, F and G are block-diagonal matrices, each comprising p matrices

$$A = \mathrm{diag}([A_1, \dots, A_p]) \tag{3.9}$$

(B, C, D, E, F and G matrices respectively). The resulting matrix has a particular structure based on the block matrices

$$T = \begin{bmatrix} D_1 & G_1F_2 & G_1E_2F_3 & G_1E_2E_3F_4 \\ C_2B_1 & D_2 & G_2F_3 & G_2E_3F_4 \\ C_3A_2B_1 & C_3B_2 & D_3 & G_3F_4 \\ C_4A_3A_2B_1 & C_4A_3B_2 & C_4B_3 & D_4 \end{bmatrix} \tag{3.10}$$

(exemplary shown for the case $p = 4$ here).

Efficient Matrix-Vector Multiplication

Depending on the dimensions of u_k, a_k, x_k, and $\hat{x}_k$, the product between an SSS matrix $T \in \mathbb{R}^{r \times v}$ with an arbitrary vector can be computed efficiently. I denote the maximum dimension of all causal and anti-causal states as d, i.e.

$$\max_k \dim(x_k) \leq d \tag{3.11}$$

and

$$\max_k \dim(\hat{x}_k) \leq d. \tag{3.12}$$

Furthermore, I assume that

$$\max_k \dim(u_k) < d \tag{3.13}$$

and

$$\max_k \dim(a_k) < d \tag{3.14}$$

(which is typically the case when looking at time-varying systems). As a result, computing the product between an SSS matrix and an arbitrary vector can be performed with $O(pd^2)$ operations (compared to $O(rv)$ in the standard case). This can be seen by looking at Equations 3.1-3.5, which can be used to compute the outcome of the matrix-vector multiplication. Therefore, we can expect a reduction for the required number of operations if d is sufficiently small. Besides the reductions for the computational costs, the storage cost also decreases to the same order of magnitude.

3.2. Neural Networks

Neural Network Architectures

In FFNNs, information is passed from from anterior layers to posterior layers. Recurrent connections are prohibited, since information is always passed forward. Other types, for example recurrent NNs, are outside the scope of this thesis.

I consider two specific types of FFNNs, namely fully connected NNs and CNNs. The latter type is usually used for image-based tasks, and may also contain a fully connected part.

Fully connected NNs got their name from the fact that each neuron in one layer is connected to all neurons in the following layer. I define fully connected NNs $N(x, W_1, \dots W_L, b_1, \dots, b_L)$ as composition of L layer mappings Λ_f for $f = 1, \dots, L$ (with $L \geq 1$)

$$N(x, W_1, \dots, W_L, b_1, \dots, b_L) = \Lambda_L(W_L, b_L, \cdot) \circ \cdots \circ \Lambda_2(W_2, b_2, \cdot) \circ \Lambda_1(W_1, b_1, x), \tag{3.15}$$

where x is the input to the NN with $x \in \mathbb{R}^m$. W_f and b_f are the weight matrices and biases which parameterize the network.

Each layer mapping Λ_f consists of a matrix-vector multiplication between the weight matrix W_f and the inputs to the layer (which are the outputs of the previous layer Λ_{f-1}), followed by adding a bias b_f and applying a (nonlinear) activation function σ_f

$$\Lambda_f(W_f, \Lambda_{f-1}, b_f) = \sigma_f(W_f \Lambda_{f-1} + b_f). \tag{3.16}$$

The activation function σ_f is applied element-wise to its inputs. The inputs for the first layer Λ_1 are given by the input layer Λ_0, which equals the inputs to the NN $\Lambda_0 = x$.

Nowadays, there are many applications for NNs in the domain of images. This includes, for example, image recognition, object detection and image segmentation. In this domain, decisions must be made based on given images. Transferred to the NN domain, this means that the inputs to the network are tensors $I \in \mathbb{R}^{r \times s \times c}$, where $r \times s$ are the dimensions of the image, and c are the number of channels in the image.

For addressing image based problems with NNs, usually CNNs are used. CNNs are another type of FFNNs with a special architecture. They typically consist of a feature extractor part and a classifier part. The feature extractor part comprises convolutional and pooling layers, which are designed to extract features from images. After the feature extractor part, the activations get reorganized into a vector, which is fed to the classifier part. Typically, the classifier part is a fully connected NN.

Convolutional layers consist of several feature maps $\Lambda_{f,j}$ for $j = 1, \dots, c_{out}$, which form the channels of the layer output Λ_f with c_{out} channels. The feature maps are computed by cross-correlating the inputs Λ_{f-1} comprising c_{in} channels with the kernel

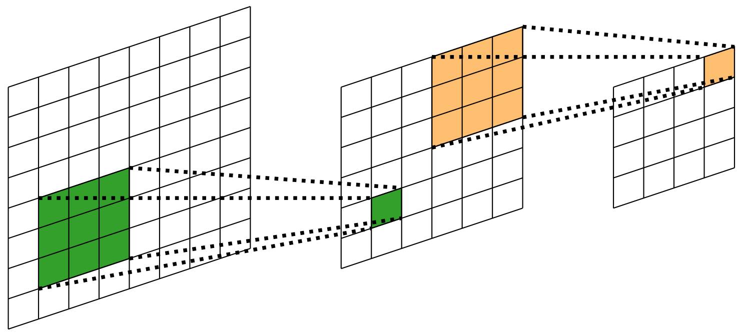

Figure 3.1: Schematic illustration of two convolutional layers with a single feature map each. Each convolutional layer extracts features using its receptive field, which is cross-correlated with the input to the layer. By stacking multiple layers on top of each other, features with different degrees of abstraction can be extracted.

of the feature map $K_{f,j} \in \mathbb{R}^{s \times r}$

$$\Lambda_{f,j} = \sigma_f \big(b_{f,j} + \sum_{t=1}^{c_{in}} K_{f,j} * \Lambda_{f-1,t} \big), \tag{3.17}$$

where $*$ is the cross-correlation operator, σ_f is the activation function, which is applied element-wise, and $b_{f,j}$ are the biases of the j^{th} feature map [65]. The dimensions of the kernel define the size of the receptive field of the feature map and are treated as hyperparameters. Additional hyperparameters are the stride of the cross-correlation operation and the padding of the inputs. The cross-correlation operator is the same as a convolutional operator with a flipped kernel, giving CNNs originally their name. However, since the cross-correlation operator is more straightforward to implement [31], it has replaced the convolution operator in most machine learning frameworks. A schematic illustration of two convolutional layers with each having a single feature map is given in Figure 3.1.

The second type of layers commonly used in CNNs are pooling layers. Pooling layers build summary statistics of nearby outputs in the previous layer. The aim is to make the representation approximately invariant to minor modifications of the input (e.g. translations) [31]. One example is the max pooling layer, which returns the maximum activation of different regions in its inputs. For that, a receptive field is moved over the inputs and for every displacement the maximum activation is determined. Pooling layers do not introduce additional trainable parameters into the NN. However, they can introduce additional hyperparameters, like for example the size or the stride of the receptive field used for the pooling operation.

The focus of this work lies on the matrix-vector multiplication in the NN. For this operation, the computational as well as storage costs scale quadratically with the size of the weight matrices. Due to the classifier part typically being a fully connected NN, this also affects modern CNN architectures. This can especially be important for the storage cost, because the weights of the convolutional layers are shared. As a result, the fully connected layers of deep CNNs typically make up the large majority of the parameters [85].

Backpropagation

NNs are *trained* to have a certain input-output behavior. For that, the training set $(X_{\text{train}}, Y_{\text{train}})$ consisting of input samples X_{train} and corresponding labels Y_{train} is used. The input set and labels each comprise m samples, which are used as examples how the inputs should be transformed into outputs. In order for the NN to match the desired transformation, the weights and biases of the network need to be adjusted. This can be done by using the backpropagation algorithm [67], which is a gradient-descent based method.

When NNs are trained using backpropagation, they are first initialized randomly, which means that random values are assigned to the weights and biases in the network. The training consists of iteratively adjusting the weights and biases in multiple training epochs. Each training epoch consists of three steps: A forward pass, a backward pass and a gradient descent step. During the forward pass of the k^{th} epoch, the outputs of the NN Y_{pred} for the samples in the training set are computed. These outputs are then compared with the desired outputs Y_{train} using a loss function $\mathcal{L}(Y_{\text{pred}}, Y_{\text{train}})$. During the backpropagation step, the loss determined by the loss function is derived with respect to the parameters of the network

$$\frac{\delta \mathcal{L}(Y_{\text{pred}}, Y_{\text{train}})}{\delta W_f^{(k)}} \tag{3.18}$$

and

$$\frac{\delta \mathcal{L}(Y_{\text{pred}}, Y_{\text{train}})}{\delta b_f^{(k)}} \tag{3.19}$$

for $f = 1, \ldots, L$ (kernel parameters in CNNs analogously). During this step, the gradients are propagated from layer to layer in the network, giving the backpropagation algorithm its name. Subsequently, the weights and biases are updated by taking a step in the negative direction of the gradient

$$W_f^{(k+1)} = W_f^{(k)} - \alpha \frac{\delta \mathcal{L}(Y_{\text{pred}}, Y_{\text{train}})}{\delta W_f^{(k)}} \text{ for } f = 1, \ldots, L \tag{3.20}$$

(other parameters analogously). The step-size α can either be fixed (usually to small values like 10^{-3}), or adapted during training (for example using a step-size optimizer like Adam [47]).

Since this procedure can be very resource intensive, often the training set is split into several mini batches. Then, one training epoch consists of subsequently updating the weights and biases of the network with respect to all mini batches.

4. Research Questions

My research questions are related to the use of SSS matrices in NNs. Here, I am interested in the benefits of using SSS weight matrices, the differences between methods used to bring the structure into the NN, and the behavior compared to weight matrices of other structure types. For each of these areas of interest, I formulate one research question, which I answer in this thesis. In order to answer the questions, I formulate hypotheses which are verified or falsified throughout the thesis.

The first research question is about the benefits of using SSS matrices as weight matrices in NNs. In contrast to other matrix structures, the structure class of SSS matrices has not been investigated in the domain of NNs yet. By answering the first question, I investigate the advantages of using this structure class in NNs.

Research Question 1 *What are the benefits of using SSS matrices as weight matrices in NNs in terms of generalization capability and resource requirements?*

I address this question by examining the following two hypotheses.
Hypothesis 1.1: NNs with SSS matrices achieve equal or better test prediction accuracy, despite having fewer trainable hyperparameters compared to standard NNs.
Hypothesis 1.2: The time needed for propagating information through the NN can be decreased by using SSS weight matrices in NNs.

In the context of the drone example introduced in Section 1.2, Hypothesis 1.1 investigates whether networks with a small number of parameters are suitable for controlling the drone. Analogously, Hypothesis 1.2 addresses the question whether SSS matrices can be used to reduce the inference time for controlling the drone. The time saved could then be used, for example, to achieve a higher control frequency. Both hypotheses are connected with each other by the trade-off on how many parameters can be spared to increase the control frequency so that the control is still flying sufficiently good and robust.

There are several ways in which structure can be introduced into the NN. For example, it can be imposed during training, or the weight matrices can be approximated with structured weight matrices after training. This raises the question how the choice of the method, that brings the structure into the network, influences the performance of the resulting NN.

Research Question 2 *Which influence does the choice of the method used to bring SSS structure into NNs have on the test accuracy?*

To answer this research question, I compare two methods for introducing structure into NNs. In the following, I refer to approaches based on training data as *data driven* methods. For example, if a NN used for controlling a drone is trained based on sensory data collected while flying on example trajectories, I would refer to this approach as data driven. In contrast, non data driven methods do not require any training data. Instead, with non data driven methods, the approach is to extract data from already trained models (for example by approximating the weight matrices of the model). Regarding the drone example, non data driven methods can be used if we already have an NN, which is able to robustly control the drone. In this case, we can analyze the given network in order to improve it by for example exploiting structure present in the given weight matrices. This leads to the following two hypotheses, which I investigate in order to answer Research Question 2.
Hypothesis 2.1: NNs with SSS weight matrices optimized using a data driven approach achieve higher test prediction accuracy than NNs whose weight matrices are approximated with SSS matrices after training.
Hypothesis 2.2: Fine-tuning NNs with approximated SSS weight matrices leads to higher test accuracy than training NNs with SSS weight matrices from scratch.

As mentioned before, there are several matrix structure classes which can be used in the domain of NNs. For the use in NNs, the most interesting structure types are those that reduce the computational resources needed to deploy the network. Usually, the reduced resource consumption is associated with a reduction in the number of parameters. Here the question arises, which influence the choice of the structure has on the performance of the network (while fixing the number of parameters in the network). Regarding the drone example, the question is if using one matrix structure in the weight matrices of the NN in favor of another structure type can lead to a more robust flying performance. This question is addressed by my third research question.

Research Question 3 *Which influence does the choice of the structure class brought into the NN have on the achieved test accuracy?*

To answer this question, I examine two hypotheses regarding the impact of the chosen structure.
Hypothesis 3.1: The test prediction accuracy of NNs with structured weight matrices approximated from trained weight matrices does not depend on the structure type if the number of parameters is the same.
Hypothesis 3.2: NNs with SSS weight matrices achieve the same prediction accuracy as NNs comprising structured weight matrices of other types when trained using gradient-descent.

5. Contributions

5.1. Structured Matrices and their Application in Neural Networks: A Survey

Authors: Matthias Kissel and Klaus Diepold
Journal: New Generation Computing
Publisher, publication year: Springer, 2023
Core publication: Yes
Bibliography Entry: [50] (`https://doi.org/10.1007/s00354-023-00226-1`)

Summary

Structured matrices can help to reduce the vast resource consumption of modern NNs. However, information about matrix structures is quite fragmented. In this paper, we give an overview over the four main matrix structure classes and provide references to research papers in which structured matrices are used in the domain of NNs. I use this overview in this thesis as a basis for my literature review and for classifying the state-of-the-art. Moreover, we present two benchmarks in the paper. First, we benchmark the error for approximating different test matrices with structured matrices of different types. Second, we compare the prediction performance of NNs in which the weight matrix of the last layer is replaced by structured matrices. I use the results of these benchmarks to answer the third research question of this thesis, concerning the effect of using different structure types in the weight matrices of NNs.

Own Contributions

- Conduct literature review to identify relevant sources and classification of the found sources into structure classes in cooperation with Prof. Diepold
- Experimental design of the benchmarks presented in the paper
- Implementation and execution of the benchmarks with subsequent discussion of the results
- Identification of research areas which are currently relevant for the field

New Generation Computing (2023) 41:697–722
https://doi.org/10.1007/s00354-023-00226-1

Structured Matrices and Their Application in Neural Networks: A Survey

Matthias Kissel[1] · **Klaus Diepold**[1]

Received: 19 December 2022 / Accepted: 21 June 2023 / Published online: 26 July 2023

Abstract

Modern neural network architectures are becoming larger and deeper, with increasing computational resources needed for training and inference. One approach toward handling this increased resource consumption is to use structured weight matrices. By exploiting structures in weight matrices, the computational complexity for propagating information through the network can be reduced. However, choosing the right structure is not trivial, especially since there are many different matrix structures and structure classes. In this paper, we give an overview over the four main matrix structure classes, namely semiseparable matrices, matrices of low displacement rank, hierarchical matrices and products of sparse matrices. We recapitulate the definitions of each structure class, present special structure subclasses, and provide references to research papers in which the structures are used in the domain of neural networks. We present two benchmarks comparing the classes. First, we benchmark the error for approximating different test matrices. Second, we compare the prediction performance of neural networks in which the weight matrix of the last layer is replaced by structured matrices. After presenting the benchmark results, we discuss open research questions related to the use of structured matrices in neural networks and highlight future research directions.

Keywords Matrix structures · Neural network · Efficient propagation · Fast inference

Matthias Kissel
matthias.kissel@tum.de

[1] TUM School of Computation, Information and Technology, Technical University of Munich, Arcisstr. 21, 80333 Munich, Bavaria, Germany

1 Introduction

1.1 Structured Matrices

When talking about structured matrices, we build on the notion of data-sparse matrices. Data sparsity means that the representation of an $n \times n$ matrix requires less than $\mathcal{O}(n^2)$ parameters. In contrast to sparse matrices, *data sparse* matrices must not contain zero entries. Instead, there is a relationship between the entries of the matrix. The simplest examples are rank 1 matrices of the form $u \cdot v^T$, for vectors $u, v \in \mathbb{R}^n$. Other easily identifiable examples of data sparse matrices are Toeplitz or Hankel matrices, which may hold $2n - 1$ parameters.

In other, less obvious cases, data sparsity implies that the entries of the respective structured matrices have an intrinsic relationship to each other. As an example for such a relationship, we can point at orthogonal matrices, which comprise $\frac{n(n-1)}{2}$ free parameters. However, orthogonal matrices have obviously $\mathcal{O}(n^2)$ parameters, which means that they do not belong to the class of data-sparse matrices.

We are particularly interested in data-sparse matrices, for which we can find efficient algorithms, for example, computing the matrix–vector product with an arbitrary vector with less than $\mathcal{O}(n^2)$ operations. There exist various matrix structures, which serve as candidates for accomplishing this goal. However, the knowledge on the subject area is quite fragmented containing many approaches originating from diverse fields. In this paper, we give an overview over the four most important structure classes. We put these classes in relation to each other, helping to reveal their boundaries and limitations. By that, we categorize the state-of-the-art in the field of structured matrices.

1.2 Computational Challenges for Neural Networks

Neural networks solve increasingly complex tasks of machine intelligence, like beating humans in the game of Go [78]. However, with increasing complexity of the problems, the complexity of the networks also increases significantly. This creates a trend toward deep networks [45, 90], which consist of a large number of layers and millions of parameters. This increase in complexity creates challenges for practical implementations, where the number of arithmetic operations grows disproportionally fast.

This trend results in the following list of technical challenges:

Training time
The training of deep neural networks can last several weeks even on modern computing architectures. For example, the training of the AlphaGo Zero network, which is able to beat the best human Go players, took 40 days (on specialized hardware) [78]. Long training times result in high costs, for example, due to high server costs. Moreover, long training periods effectively hinder to adapt quickly to new data.

Inference time
The more operations need to be performed in order to compute the output of a neural network, the more time is needed for the computation. Thus, the inference time directly

scales with the number of operations in the neural network (neglecting parallelization capabilities). If the inference takes too long, the applicability of a neural network is restricted to certain applications, where fast inference is not essential. For example, the AlphaGo Zero network needed specialized hardware to be able to answer with reasonably low response time, which is required for playing a game of Go. In the case of AlphaGo Zero, 4 tensor processing units were used in order to perform inference in at most 5 s, and previous versions were distributed on up to 176 GPUs for calculating the next move in real time [78].

Memory requirements
Large neural networks consist of many parameters, which need to be stored. For example, the popular pre-trained ResNet50 [45] network needs 98MB memory space (provided by the keras project[1]). This is by far not the upper limit—there are much bigger architectures available and in use. The required memory capacity can be problematic for resource constraint devices, such as mobile devices, smartphones or microcontrollers. For standard computers (PCs), the amount of memory required to load the whole model into RAM may also be prohibitive.

Memory bandwidth requirements
Besides the large memory needed to store the parameters of a given deep neural network, it is also an issue to provide the memory bandwidth necessary to facilitate fast learning or fast inference. Indeed, it has been shown that for deep neural networks memory access is the main bottleneck for processing [82]. Therefore, significant processing speedups can be achieved by optimizing the memory access to reduce bandwidth [46].

Power consumption
As the amount of operations for performing training or during inference increases, the power consumption also increases. Again, the increasing power consumption is challenging for mobile devices or, more generally, for all battery-driven systems. Besides the costs arising with increased power consumption, neural networks might thus contribute to today's climate change. For example, training big natural language processing models including hyper parameter search can produce up to twice the amount of CO_2 produced by an average American within one year [81]. Therefore, we are usually interested in reducing the power requirements.

1.3 Goals and Organization

Numerous researchers have contributed to mitigate the aforementioned problems. For example, a survey on increasing the efficiency of neural networks is given by Sze et al. [82]. In this paper, we focus on approaches using structured matrices in the domain of neural networks, which has the potential to overcome all mentioned problems.

We see two main advantages of using structured matrices in neural networks to save resources compared to other approaches. First, for many structures, it is possible to train the neural network end-to-end. This means that conventional, well-tested training algorithms such as backpropagation can be used for training. In comparison,

[1] https://keras.io/.

most methods to save resources in neural networks start only after the training, which may lead to worse results. Second, in contrast to the common mindset that resource savings in neural networks always lead to performance losses, we assume that the choice of the right structure can even *improve* the performance. This is the case if the chosen structure fits the problem, and thus the search space for the weights of the neural network is restricted in a meaningful way.

Contribution
Our main contribution is to give an overview over the most important matrix structure classes, and to present two benchmarks comparing the classes. We briefly introduce each structure class, and mention efficient algorithms. For each class, we analyze the computational requirements for computing the matrix–vector product, which plays a major role in neural networks. Moreover, we review approaches where each structure has been used in the domain of neural networks. Through this, our survey offers a starting point to choosing the right structure for a given problem.

Organization
The paper is organized as follows—we first introduce the four main structure classes which we identified from literature, namely semiseparable matrices, matrices of low displacement rank, hierarchical matrices, and products of sparse matrices. Subsequently, we set the structure classes into relation to each other, showing their boundaries. We present two benchmarks comparing the structure classes. The first benchmark investigates the error for approximating different test matrices. The second benchmark compares the prediction performance of neural networks, in which the weight matrix of the last layer is replaced by a structured matrix. In the following section, we point out open research questions and future research directions for using structured matrices in the domain of neural networks. Finally, we summarize our findings and draw a conclusion.

2 Classes of Structured Matrices

2.1 Semiseparable Matrices

The first notion of semiseparable matrices [87] appears in work published in 1937 by Gantmakher and Krein [33, 86]. Since then, there has been a number of publications and generalizations of results to the class of semiseparable matrices [86]. The motivation for research about semiseparable matrices originates from various application domains for computational science and engineering, such as for example time-varying system theory [22], where the matrices appear in the context of simulating physical phenomena and systems. The most prominent representatives in this class are tridiagonal matrices and other banded matrices along with their inverses.

Definition
We focus on the definition of sequentially semiseparable matrices [22]. A sequentially semiseparable matrix T has a block structure based on the matrices A_k, B_k, C_k, D_k, E_k, F_k and G_k

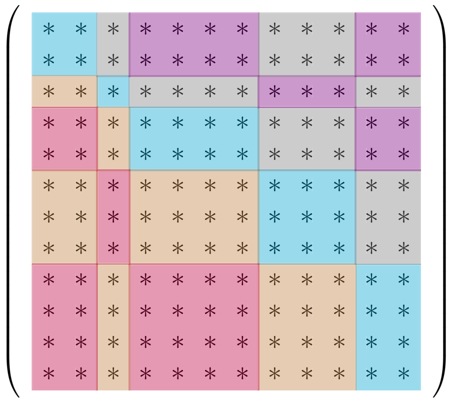

Fig. 1 Schematic Illustration of the partitioning of a sequentially semiseparable matrix. The rectangular shapes of the submatrices illustrate that the input, output, and state dimensions associated with the sequentially semiseparable matrix can change between timesteps

$$T_{i,j} = \begin{cases} D_i & \text{for} \quad i = j, \\ C_i A_{i-1} \dots A_{j+1} B_j & \text{for} \quad i < j, \\ G_i E_{i+1} \dots E_{j-1} F_j & \text{for} \quad i > j. \end{cases} \tag{1}$$

This structure arises in the transfer matrix of a time-varying system with state equations

$$x_{k+1} = A_k x_k + B_k u_k, \tag{2}$$

$$\hat{x}_k = E_k \hat{x}_{k+1} + F_k u_k, \tag{3}$$

$$a_k^{(1)} = C_k x_k + D_k u_k, \tag{4}$$

$$a_k^{(2)} = G_k \hat{x}_{k+1}, \tag{5}$$

and

$$a_k = a_k^{(1)} + a_k^{(2)}, \tag{6}$$

which reveals why this structure is closely related to the theory of time-varying systems. In the domain of time-varying systems, x_k refers to the state of the causal part of the system at timestep k ($\hat{x}_k$ to the anti causal part respectively), u_k are the inputs to the system at timestep k and a_k are the outputs respectively. Note that the dimensions of the A_k, B_k, C_k, D_k, E_k, F_k and G_k might change for different timesteps, which reflects the fact that the state, the input as well as the output dimension might change over time. This structure leads to a sequentially partitioning of the matrix as exemplary illustrated in Fig. 1. There are also other definitions for semiseparable matrices [87], for example, for quasiseparable matrices. *A matrix S is called a quasiseparable matrix if all the subblocks taken out of the strictly lower triangular part of the matrix (respectively the strictly upper triangular part) are of rank 1.*

Special Structures

The class of semiseparable matrices can be seen as collection of slightly different definitions for semiseparability [87], such as sequentially semiseparable, generator-representable semiseparable, semiseparable plus diagonal and quasiseparable matrices. For example, the class of semiseparable plus diagonal matrices extends the class of semiseparable matrices by adding a diagonal to the semiseparable matrix. The set of generator-representable semiseperable matrices includes all matrices, where the upper and lower triangular parts are coming from a rank 1 matrix (in contrast to general semiseparable matrices, where the sub-blocks of the lower or upper triangular matrix may come from different rank 1 matrices). Another class of semiseparable matrices are hierarchically semiseparable matrices [87], which are closely connected with the class of hierarchical matrices introduced in Sect. 2.3. Examples for special matrices belonging to the class of semiseparable matrices are band matrices [24] or their inverses [75].

Efficient Algorithms
By exploiting the semiseparable structure, the number of operations for computing the matrix vector product can usually be reduced to $\mathcal{O}(nd^2)$, where d is the maximum state dimension

$$d = \max_k(\max(\dim(x_k), \dim(\hat{x}_k))). \tag{7}$$

This reduction comes from an efficient computational scheme exploiting the sequential structure, which is based on systematically using intermediate results of matrix–vector products of the submatrices. Depending on the structure at hand, there are numerous other fast algorithms available, which may not apply for the general class of semiseparable matrices. A rigorous historic overview of the results found for the class of semiseparable matrices is given by Vandebril et al. [86]. For example, there is a fast algorithm for calculating the inverse of a generator representable plus diagonal semiseparable matrix [23].

Application to Neural Networks
Kissel et al. [51, 52, 52] analyzed the effect of using sequentially semiseparable weight matrices in neural networks. They introduced the *Backpropagation through states* algorithm [51], which can be used to train neural networks with sequentially semiseparable weight matrices. Moreover, they showed how trained weight matrices can be approximated with sequentially semiseparable matrices [50, 52]. Their experiments showed that depending on the task at hand, neural networks with sequentially semisperable weight matrices are able to outperform their standard counterparts in terms of generalization performance [51].

2.2 Matrices of Low Displacement Rank

The class of matrices with Low Displacement Rank (LDR) [67] unifies the probably most prominent matrix structures, including Toeplitz, Hankel, Vandermonde and Cauchy matrices. The idea of a displacement representation originates from modeling stochastic signals, which may exhibit mild forms of non-stationarity, leading to notions such as Toeplitz-like or Hankel-like displacements [67].

Definition
For analyzing the displacement rank [67] of a matrix M, either the displacement operators of the Sylvester type

$$L(M) = \nabla_{A,B}(M) = AM - MB, \tag{8}$$

$$\begin{pmatrix} u_0 & v_0 & v_1 & v_2 & v_3 & v_4 \\ u_1 & u_0 & v_0 & v_1 & v_2 & v_3 \\ u_2 & u_1 & u_0 & v_0 & v_1 & v_2 \\ u_3 & u_2 & u_1 & u_0 & v_0 & v_1 \\ u_4 & u_3 & u_2 & u_1 & u_0 & v_0 \\ u_5 & u_4 & u_3 & u_2 & u_1 & u_0 \end{pmatrix}$$

(a) Toeplitz Matrix $M_T(u, v)$

$$\begin{pmatrix} v_4 & v_3 & v_2 & v_1 & v_0 & u_0 \\ v_3 & v_2 & v_1 & v_0 & u_0 & u_1 \\ v_2 & v_1 & v_0 & u_0 & u_1 & u_2 \\ v_1 & v_0 & u_0 & u_1 & u_2 & u_3 \\ v_0 & u_0 & u_1 & u_2 & u_3 & u_4 \\ u_0 & u_1 & u_2 & u_3 & u_4 & u_5 \end{pmatrix}$$

(b) Hankel Matrix $M_H(u, v)$

$$\begin{pmatrix} 1 & u_0 & u_0^2 & u_0^3 & \dots & u_0^{n-1} \\ 1 & u_1 & u_1^2 & u_1^3 & \dots & u_1^{n-1} \\ 1 & u_2 & u_2^2 & u_2^3 & \dots & u_2^{n-1} \\ 1 & u_3 & u_3^2 & u_3^3 & \dots & u_3^{n-1} \\ 1 & u_4 & u_4^2 & u_4^3 & \dots & u_4^{n-1} \\ 1 & u_5 & u_5^2 & u_5^3 & \dots & u_5^{n-1} \end{pmatrix}$$

(c) Vandermonde Matrix $M_V(u)$

$$\begin{pmatrix} \frac{1}{u_0-v_0} & \frac{1}{u_0-v_1} & \frac{1}{u_0-v_2} & \frac{1}{u_0-v_3} & \frac{1}{u_0-v_4} \\ \frac{1}{u_1-v_0} & \frac{1}{u_1-v_1} & \frac{1}{u_1-v_2} & \frac{1}{u_1-v_3} & \frac{1}{u_1-v_4} \\ \frac{1}{u_2-v_0} & \frac{1}{u_2-v_1} & \frac{1}{u_2-v_2} & \frac{1}{u_2-v_3} & \frac{1}{u_2-v_4} \\ \frac{1}{u_3-v_0} & \frac{1}{u_3-v_1} & \frac{1}{u_3-v_2} & \frac{1}{u_3-v_3} & \frac{1}{u_3-v_4} \\ \frac{1}{u_4-v_0} & \frac{1}{u_4-v_1} & \frac{1}{u_4-v_2} & \frac{1}{u_4-v_3} & \frac{1}{u_4-v_4} \end{pmatrix}$$

(d) Cauchy Matrix $M_C(u, v)$

Fig. 2 Schematic drawings of the most popular low displacement rank special cases. The displacement structure can be seen in all four matrices: The same values (or modified values) appear in different positions of the matrices

or of the Stein type

$$L(M) = \Delta_{A,B}(M) = M - AMB, \tag{9}$$

can be used. A and B are operator matrices defining the displacement. A matrix has low displacement rank if the displacement matrix $L(M)$ is of low rank. There exists an abundance of possible definitions for the displacement operators and hence this class is quite big.

Efficient Algorithms

There exist efficient algorithms for certain tasks given that the rank of the displacement matrix $L(M)$ is small. This is based on the assumption that the matrix can be *compressed* using the displacements, and that operations can be performed faster using the compressed version. The original matrix can be *recovered (decompressed)* from the displacements. The overall operation scheme can be described as $Compress \rightarrow Operate \rightarrow Decompress$. By exploiting this scheme, inter alia the matrix–vector multiplication can be made more efficient. This leads, for example, to $\mathcal{O}(n\log(n))$ operations for Toeplitz and Hankel matrices, and $\mathcal{O}(n\log^2(n))$ operations for Vandermonde and Cauchy matrices in order to compute the matrix–vector product of a matrix $M \in \mathbb{R}^{n\times n}$ with an arbitrary n-dimensional vector [80].

Table 1 Operator matrices and rank of the corresponding displacements for Toeplitz, Hankel, Cauchy and Vandermonde matrices

Matrix type	Symbol	A	B	Rank$(\nabla_{A,B}(M))$
Toeplitz matrix	$M_T(u,v)$	Z_1	Z_0	≤ 2
Hankel matrix	$M_H(u,v)$	Z_1	Z_0^T	≤ 2
Vandermonde matrix	$M_V(u)$	$D(u)$	Z_0	≤ 1
Cauchy matrix	$M_C(u,v)$	$D(u)$	$D(v)$	≤ 1

Operator matrices are given with respect to the Sylvester displacement (Eq. 8)

Special Structures
The most popular special members of this structure class are Toeplitz, Hankel, Vandermonde and Cauchy matrices (depicted in Fig. 2). For these special cases, the operator matrices are based on f-circulant matrices

$$Z_f = \begin{pmatrix} 0 & & & f \\ 1 & \ddots & & \\ 0 & \ddots & \ddots & \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{10}$$

or diagonal matrices $D(u)$ defined by the vector u. For example, the operator matrices for a Toeplitz matrix $M_T(u,v)$ as depicted in Fig. 2 with respect to the Sylvester displacement operator are $A = Z_1$ and $B = Z_0$. Hence, the displacement matrix for a Toeplitz matrix $L(M_T(u,v))$ is given by

$$L(M_T(u,v)) = \nabla_{Z_1,Z_0}(M_T(u,v)) = Z_1 M_T(u,v) - M_T(u,v) Z_0. \tag{11}$$

For all Toeplitz matrices $M_T(u,v)$, the rank of the displacement matrix $L(M_T(u,v))$ fulfills

$$\text{rank}(L(M_T(u,v))) \leq 2. \tag{12}$$

The displacement operators for the other special cases are given in Table 1.

Application to Neural Networks
There are several approaches in literature using matrices of low displacement rank in neural networks. The most prominent example is the Convolutional Neural Network (CNN) architecture [66], which is based on sparse Toeplitz Matrices. Convolutional Neural Networks are due to their efficiency and prediction performance the number one choice in machine learning tasks related to images nowadays [45, 53, 79]. In CNNs, the structure is usually encoded implicitly by the connections between the neurons. There are also interesting approaches for improving traditional CNNs. For example, Quaternion CNNs [34, 68, 95] perform operations on images represented in the quaternion domain, which enables them to outperform standard real-valued CNNs on several benchmark tasks. Other approaches focused on matrix structures apart from neural network architectures. Liao and Yuan proposed to use matrices with

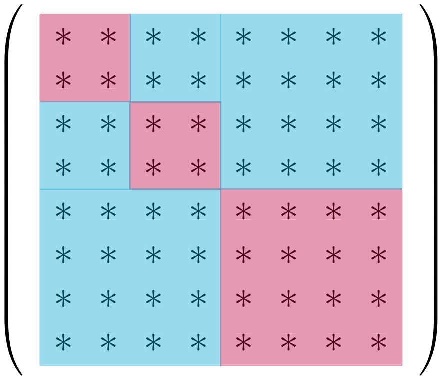

Fig. 3 Schematic Illustration of a (very simple) hierarchical matrix. The cyan parts are low-rank submatrices (admissible blocks), and the purple parts are full-rank submatrices (inadmissible blocks)

a circulant structure in Convolutional Neural Networks [60] and Cheng et al. replaced the linear projections in fully connected neural networks with circulant projections [13]. Appuswamy et al. [4] combined the efficient weight representation used in neuromorphic hardware with block Toeplitz matrices arising in discrete convolutions, which resulted in a family of convolution kernels that are naturally hardware efficient. It has also been proposed to replace weight matrices with general matrices of low displacement rank in neural networks. For example, Sindhwani et al. [80] used Toeplitz-like weight matrices, which include inter alia circulant matrices as well as Toeplitz matrices and their inverses. Moreover, Thomas et al. [84] introduced a class of low displacement rank matrices for which they trained the operators as well as their low-rank components in the neural network. Other works investigate the theoretical properties of neural networks with weight matrices of low displacement rank. For example, Zhao et al. [94] inter alia showed that the universal approximation theorem holds for these networks. Another proof showing that the universal approximation theorem holds for neural networks comprising Toeplitz or Hankel weight matrices is given by Liu et al. [61]. Their approach can be viewed as a Toeplitz-, Hankel-, or LU-based decomposition of neural networks. In particular, they present two proofs for the universal approximation theorem: One for neural networks with fixed depth and arbitrary width, and a second for neural networks with fixed width and arbitrary depth.

2.3 Hierarchical Matrices

Hierarchical matrices ($\mathcal{H}$-matrices) are based on the principle, that even if the overall matrix does not have a low rank, there might still be low-rank sub-blocks in the matrix. Therefore, the idea is to partition a matrix into sub-matrices using suitable (potentially complex) index sets and exploit the low-rank structure of the sub-matrices in this decomposition.

Definition

$\mathcal{H}$-matrices are defined by block cluster trees [9, 41, 43]. The block cluster tree decomposes the matrix into admissible and non-admissible blocks. Being admissible means that the regarded block has a low-rank structure, and therefore can be decomposed into two matrices with at most rank r (with r being smaller than the dimensions of the block). The overall aim is to find a block cluster tree for a given matrix, such that large parts of the matrix can be approximated by low-rank matrices (and still be close to the original matrix). In Fig. 3, an example partitioning of a hierarchical matrix is depicted. In order to determine the block cluster tree, first the row and column indices of the regarded matrix are organized in cluster trees, i.e., set decomposition trees for the row and column index sets of the matrix. This can, for example, be done by geometric bisection or regular subdivision. Based on these cluster trees, the block cluster tree can be defined by forming pairs of clusters on the cluster trees recursively. The number of leaves in the block cluster tree determines the complexity for arithmetic operations. Therefore, while constructing the block cluster tree, it is desirable to ensure that blocks become admissible as soon as possible. Using these building blocks, $\mathcal{H}$-matrices are defined as follows [43]. Let $L \in \mathbb{R}^{I \times I}$ be a matrix and $\mathcal{T}_{I \times I}$ a block cluster tree for L consisting of admissible and non-admissible leaves. L is called $\mathcal{H}$-matrix of blockwise rank r, if for all admissible leaves $B \in \mathbb{R}^{\tau \times \sigma}$ defined by $\mathcal{T}_{I \times I}$

$$\operatorname{rank}(B) \leq k, \tag{13}$$

with $k \in \mathbb{N}$.

Efficient Algorithms
There are fast algorithms for the different sub-classes and special forms of this structure class. Moreover, for general $\mathcal{H}$-matrices, there is a fast algorithm for matrix–vector multiplication ($\mathcal{O}(kn \log(n))$ under moderate assumptions) [41, 43]. Efficient algorithms for arithmetic operations with $\mathcal{H}$-matrices exploit the fact, that the matrix is sub-divided into admissible and non-admissible smaller block-matrices. Based on this decomposition, arithmetic operations can be conducted faster by exploiting the low-rank structure of admissible blocks. The overall result can then be obtained by combining the results from the sub-blocks.

Special Structures
The class of $\mathcal{H}$-matrices unifies several other structures based on hierarchical decompositions. These classes include [2] hierarchically off-diagonal low-rank matrices (HOLDR) [3], hierarchically semi-separable matrices (HSS) [11, 87], $\mathcal{H}^2$-matrices [41, 42], and matrices based on the fast multipole method (FMM) [5, 6, 18, 30, 39, 40]. The relationships of the subclasses to each other as well as their separation from each other are described in [3].

Application to Neural Networks
Fan et al. [27] proposed to use hierarchical matrices in neural networks, which results in a multiscale structure inside the neural network. Later, they extended their approach to $\mathcal{H}^2$-matrices, which led to comparable results as with their original approach, but reduced number of parameters. Chen et al. [12] proposed to approximate the Generalized Gauss–Newton Hessian by a hierarchical matrix, which can be used during training, for analyzing neural networks, estimating learning rates or other applications. Hierarchical matrix approaches have also been used to analyze and compress trained neural networks. For example, Ithapu used a multi-resolution matrix factorization to

analyze the inter-class relationships of deep learning features [48]. Wu et al. [89] applied the Hierarchical Tucker decomposition [38] to neural networks in order to compress fully connected layers as well as convolutional kernels. They argued that the hierarchical matrix format obtained by the Hierarchical Tucker decomposition has advantages for compressing weight matrices in fully connected layers compared to the Tensor Train decomposition, which has been used before. Another approach is to use wavelets in neural networks [20, 25, 26, 32, 49, 71, 74, 93]. The resulting networks are called *wavelet networks* and make the time-frequency zooming property of wavelets usable in neural networks [49]. Wavelet networks are often constructed from multiresolution analysis or multiresolution approximation [25, 26, 49].

2.4 Products of Sparse Matrices

The structure classes presented in the previous sections represent data-sparse matrices. In contrast, the focus of this section are products of *sparse* matrices. While data-sparse matrices may be full matrices, i.e., all n^2 matrix entries are different from zero, we talk of sparse matrices if the matrices only contain few non-zero entries (for example, $\mathcal{O}(n)$ non-zero entries) [76]. This is an extremely important class of matrices with numerous applications and a long tradition. Exploiting the zero entries directly leads to faster algorithms for several arithmetic operations, since operations can potentially be omitted. This class is somewhat different then the ones mentioned before, as this type of sparse structure does not lend itself well for an algebraic characterization.

The product of sparse matrices is not sparse in general. Therefore, even many dense matrices can be represented as product of sparse matrices. For well known fast linear transforms, such as the Fast Fourier Transform [15], the Discrete Wavelet Transform [62] or the Hadamard transform [77], there is a structured representation as product of sparse matrices [1, 55]. In fact, the notion of sparsity and structure in linear maps seems to be fundamentally linked [16, 19]. It follows, that all efficient matrix–vector multiplication algorithms can be factorized into products of sparse matrices. The conclusion from these results is [17] that all forms of structure are captured by the representation of linear maps as product of sparse matrices (supported by results from arithmetic circuit theory [10]).

Definition
Sparse Matrices comprise only few nonzero elements [76]. This definition is somewhat vague, but in general the resulting fast algorithms are faster the fewer nonzero entries the matrix has. An example of a sparse matrix is depicted in Fig. 4. The sparsity pattern of a sparse matrix can either be structured (i.e., the nonzero elements are distributed following a regular pattern) or unstructured (with irregularly located nonzero entries). There are different storage schemes which can be used to store sparse matrices. Selecting the right storage scheme is crucial for implementing fast algorithms and depends on the application at hand (more specifically the arithmetic operations which should be performed with the sparse matrix as well as the sparsity pattern at hand). Popular storage scheme examples are the coordinate, compressed sparse row as well as the compressed sparse column matrix format. For example, the coordinate format consists of three arrays. The first array contains the values of the nonzero entries in

Fig. 4 Schematic example of a sparse matrix. Most of the entries are zero. The few non-zero elements are distributed without (obvious) regularity within the matrix

$$\begin{pmatrix} * & 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ * & 0 & * & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & * \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 \end{pmatrix}$$

the matrix, whereas the second and third array contain the row and column indices of the positions of these values in the matrix respectively.

Efficient Algorithms

Bounds on the complexity of efficient algorithms for sparse matrices depend on the number of non-zero elements in the matrix as well as the pattern of their distribution. Depending on the number of non-zero entries, there are fast algorithms for computing the matrix vector product. This does also apply for the product of sparse matrices, such that the number of operations for multiplying the product of sparse matrices with an arbitrary vector are proportional to the number of nonzero elements in the sparse matrices [16]. Fast algorithms for sparse matrix vector multiplication might suffer from several memory accessing problems [37]. This includes for example the irregular memory access for the vector with which the sparse matrix is multiplied [83] or the indirect memory references in the sparse matrix (due to the fact that only the non-zero elements of the matrix are stored) [73]. Since these problems can have a significant influence on the performance of considered arithmetic operations with sparse matrices, there have been several approaches proposed to overcome these problems [28, 35, 47, 72, 85] or tune sparse matrices for specific hardware [7, 29, 64].

Special Structures

A special form of sparse matrices are Butterfly matrices [59, 69], which encode the recursive divide-and-conquer structure of the Fast Fourier Transform [17]. Butterfly matrices are composed as a product of butterfly factor matrices. Kaleidoscope matrices [17], in turn, are the product of butterfly matrices. Dao et al. proposed Kaleidoscope matrices, because in general, it is difficult to find the best sparsity pattern for the sparse matrix factorization (since this is a discrete, non-differentiable search problem). They showed that Kaleidoscope matrices have a similar expressivity as general products of sparse matrices and that various common structured linear transforms lie within this structure class.

Application to Neural Networks

Sparsity has probably been the first structure applied to neural networks. Obtaining sparse weight matrices has for example been addressed by Hassibi and Stork [44] and Le Cun et al. [57]. Their approaches used information from second-order derivatives in order to remove unimportant weights from the network. More recent work uses group lasso regularization for structured sparsity learning [88], pruning techniques [8], hand-tuned heuristics [21] or obtain sparse neural networks by chance [31]. Products of sparse matrices have also been used in neural networks. In Butterfly networks [1, 58], the inputs to a neural network are connected to the outputs of the network using the butterfly structure. It has been shown, that the regular Convolutional Neural Network architecture is a special Inflated-Butterfly-Net (where inflated means that there are dense cross-channel connections in the network) [91]. Moreover, Li et al. [58] showed that the approximation error of Butterfly networks representing the Fourier kernels exponentially decays with increasing network depth. Dao et al. [16] also incorporated Butterfly matrices into end-to-end Machine Learning pipelines and showed that they were able to recover several fast algorithm such as the Discrete Fourier Transform or the Hadamard Transform. To overcome the non-differentiable search problem of finding the right sparsity pattern, Dao et al. [17] proposed to use Kaleidoscope matrices in neural networks. By that, the optimization problem remains differentiable while having similar expressiveness as general sparse matrix products. Giffon et al. [36] showed that replacing weight matrices in deep convolutional neural networks by products of sparse matrices can yield a better compression-accuracy trade-off than other popular low-rank-based compression techniques. Their approach is based on the algorithm proposed by Magoarou et al. [55], which finds a sparse matrix product approximation of a given matrix using projected gradient steps.

3 Relations and Comparison

3.1 Structure Classes Overview and Boundaries

After introducing the four main structure classes, we give an overview over the sub-classes, which are contained in the main structure classes. Moreover, we show that the boundaries between the structure classes are not strict, which means that some matrices can be represented in the methodology of different structure classes.

We consider the four structure classes presented in the previous chapters as the main classes of structured matrices. These classes can be used to categorize particular matrix structures which can be found in literature. Since research about structured matrices is fragmented and approaches originate from different fields, there are sub-classes which are special cases of the four main structure classes. The relations of these sub-classes are depicted in Fig. 5.

Even though the four main structure classes are based on different mathematical concepts, there are still matrix classes that can be efficiently represented in multiple structure frameworks. Low-rank matrices are an example of this. These can be represented as semiseparable matrices (since the blocks taken out of a low-rank matrices are again of low rank), hierarchical matrices (by decomposing the whole matrix into a single admissible block), as well as matrices with low displacement rank [84]. Moreover, a rank r matrix $A \in \mathbb{R}^{n \times n}$ with $A = EP^T$ can straightforwardly be represented by a product of two sparse matrices $A = VM$ with $V, M \in \mathbb{R}^{n \times n}$ by setting the first r columns of V to E (and the first r rows of M to P^T respectively).

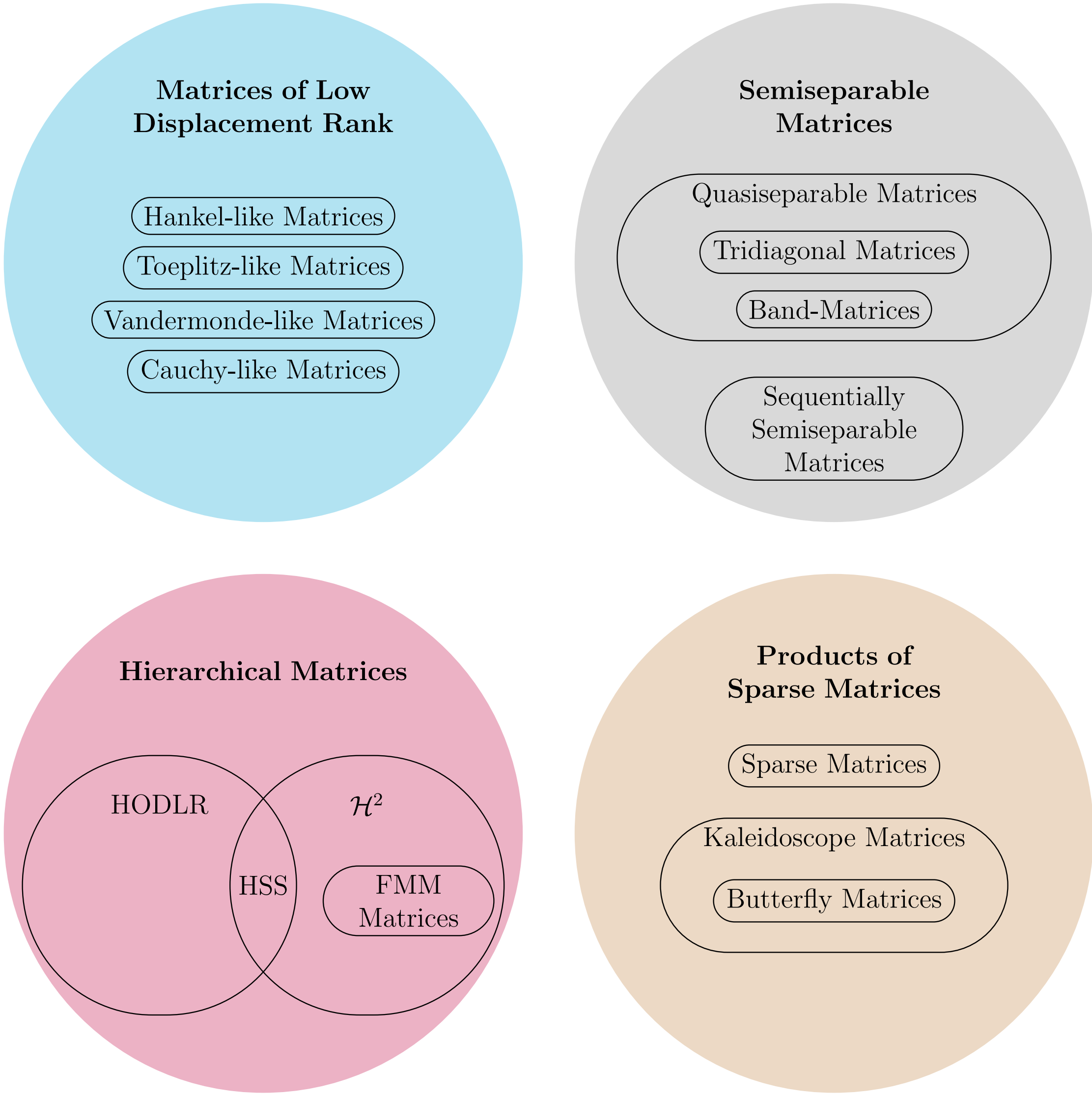

Fig. 5 Overview over the four main structure classes and structure sub-classes which they contain. The four main classes generalize concepts and approaches of special structure classes, which originated from different fields. The part about hierarchical matrices is redrawn after [2]

3.2 Benchmark: Test Matrix Approximation

One use case is that an arbitrary matrix is given, which is to be approximated with a structured matrix. If the approximated matrix is sufficiently close to the original matrix (in a metric suitable for the problem), then the original matrix can be replaced by the structured matrix. Thus, memory and potentially computational resources can be saved. In the domain of neural networks, this means that a weight matrix from a trained network is investigated to check if it possesses a certain structure. If a structure is (approximately) present, then the original weight matrix can be replaced with the new weight matrix represented in the structured matrix framework. The predictions of the neural network are then ideally similar to those before the modification, but memory and computational resources are saved.

Which structure is suited best for approximation depends on the task at hand as well as the selected metric. In this section, we give an overview over the approximation capabilities of the structure classes for different test matrices. We use the Frobenius norm as a metric for how close the approximated matrix is to the original, since this has been found to be a good surrogate for comparing weight matrices [52]. With our benchmark, we aim to give a notion in which structure classes are particularly suitable for approximating certain matrix types. However, this cannot be seen as a conclusive assessment that one structure class is always preferable to another. The choice of the right structure class still depends on the task and context at hand.

We use the following test matrices in our benchmark:

- Random Matrices (with randomly uniform distributed entries in the range $[-1, 1[$)
- Orthogonal Matrices
- Low Rank Matrices
- Matrices with linearly distributed singular values (in the interval $[0.1, 1.0]$).
- Sequentially Semiseparable Matrices (with statespace dimension set to 5)
- Products of Sparse Matrices (comprising 3 matrices each with 90% sparsity respectively)
- Hierarchical Matrices (with geometrically inspired block cluster trees as introduced in [42] with $\eta = 0.5$)
- Matrices with low displacement rank (Toeplitz, Hankel and Cauchy matrices)
- Weight matrices from Imagenet-pretrained vision models provided by PyTorch [70] (GoogleNet, InceptionV3, MobilenetV2, and Resnet18)

For each of the test matrix classes, we instantiate 3 matrices of shape 300×300 (except for the weight matrices taken out of the vision models), and approximate them using structured matrices of the presented classes. The code used for running our experiments and our test matrices (together with the scripts for generating them) are available on GitHub.[2]

For approximating the test matrices with sequentially semiseparable matrices, we use the approach described in [52] (performing a hyperparameter search for different number of stages), using the TVSCLib[3] implementation. Also, the approximation for products of sparse matrices is based on the approach presented in [52], which is in turn based on an algorithm proposed by Magoarou and Gribonval [55]. We treat the number of sparse factors as well es the sparsity distribution across the factors as hyperparameters, for which we perform a search. Our implementation for the $\mathcal{H}$-matrix approximation uses a greedy approach for assigning low-rank components to the leaf nodes of a block cluster tree. The block cluster tree is treated as hyperparameter, where we compared the admissibility criterion from Hackbusch and Börm [42] (for different values of η) with the approach of building block cluster trees with equally distributed low-rank patches of same size. For approximation with matrices of low displacement rank, we try multiple approaches. First, we investigate the approach presented in [52], which finds an approximation based on gradient-descent updates for the displacements as well as the operator matrices. Second, we employ a direct approximation scheme

2 https://github.com/MatthiasKi/structurednets.

3 https://github.com/MatthiasKi/tvsclib.

using fixed operator matrices for Toeplitz-like matrices inspired by Sindhwani et al. [80]. After applying the operator matrices, we find the truncated displacements by performing a Singular Value Decomposition (SVD) on the original displacements. We also show the approximation result for low-rank matrices as a baseline. This approximation is also based on the SVD.

As expected, the approximation error becomes smaller if more parameters are used for approximating the given matrix. Moreover, the approximation algorithms perform particularly good if the investigated matrix has the structure which is used by the approximation approach. For the methods we compared, the approximation approach of using products of sparse matrices resulted in consistently good results for all test matrices. This supports results from Dao et al. [17], stating that the structure class of products of sparse matrices is very powerful for approximating structured transforms. The results of our benchmark are depicted in Fig. 6.

For the approximated weight matrices of PyTorch vision models, we draw a similar conclusion. The products of sparse matrices achieved the best approximation results. This is in line with the findings in [52], where this observation has already been made for smaller weight matrices. For the considered weight matrices, using $\mathcal{H}$-matrices for approximation does not seem to provide much advantage over our baseline, low-rank matrices. In all cases considered, both produce similar approximation results. The approximation with sequentially semiseparable matrices led to the worst results. This was also observed in earlier experiments with smaller weight matrices [52].

We did not include the results for using matrices of low displacement rank in the plots for two reasons. First, the methods given in literature refer to square matrices, which renders them inapplicable for the considered weight matrices. This is not a general limitation, since the framework of matrices with low displacement rank is also applicable to non-square matrices [84]. However, the given algorithms for using matrices of low displacement rank, for example, for recovering a matrix from its displacements, cannot trivially be extended to non-square matrices. Second, the approach introduced by Kissel et al. [52] for approximating square weight matrices using matrices of low displacement rank is only practically usable for small matrices. This is, because the algorithm consumes too much memory and computing resources when the matrices are large (which is the case in our benchmark). Using less sophisticated approaches with fixed operator matrices (for example for Toeplitz-like or Hankel-like matrices) resulted in bad approximation results for all test matrices, except for the ones with the corresponding structure. Therefore, we conclude that the design of practically usable algorithms for the approximation of low displacement rank matrices is still an open task. However, note that apart from *approximating* given matrices, there are efficient algorithms for *training* (square) weight matrices with low displacement rank from scratch [80, 84].

Note that the approximation algorithms used in our benchmark are subject to ongoing research, and for each class there is still a lot room for improvement. Our goal was to show a fair comparison in which the hyperparameters of the individual approaches were tuned with comparable effort. Therefore, it is totally possible that improving the approximation algorithm for one of the structure classes (or developing better heuristics for finding hyperparameters) might render it superior to all other classes in the future.

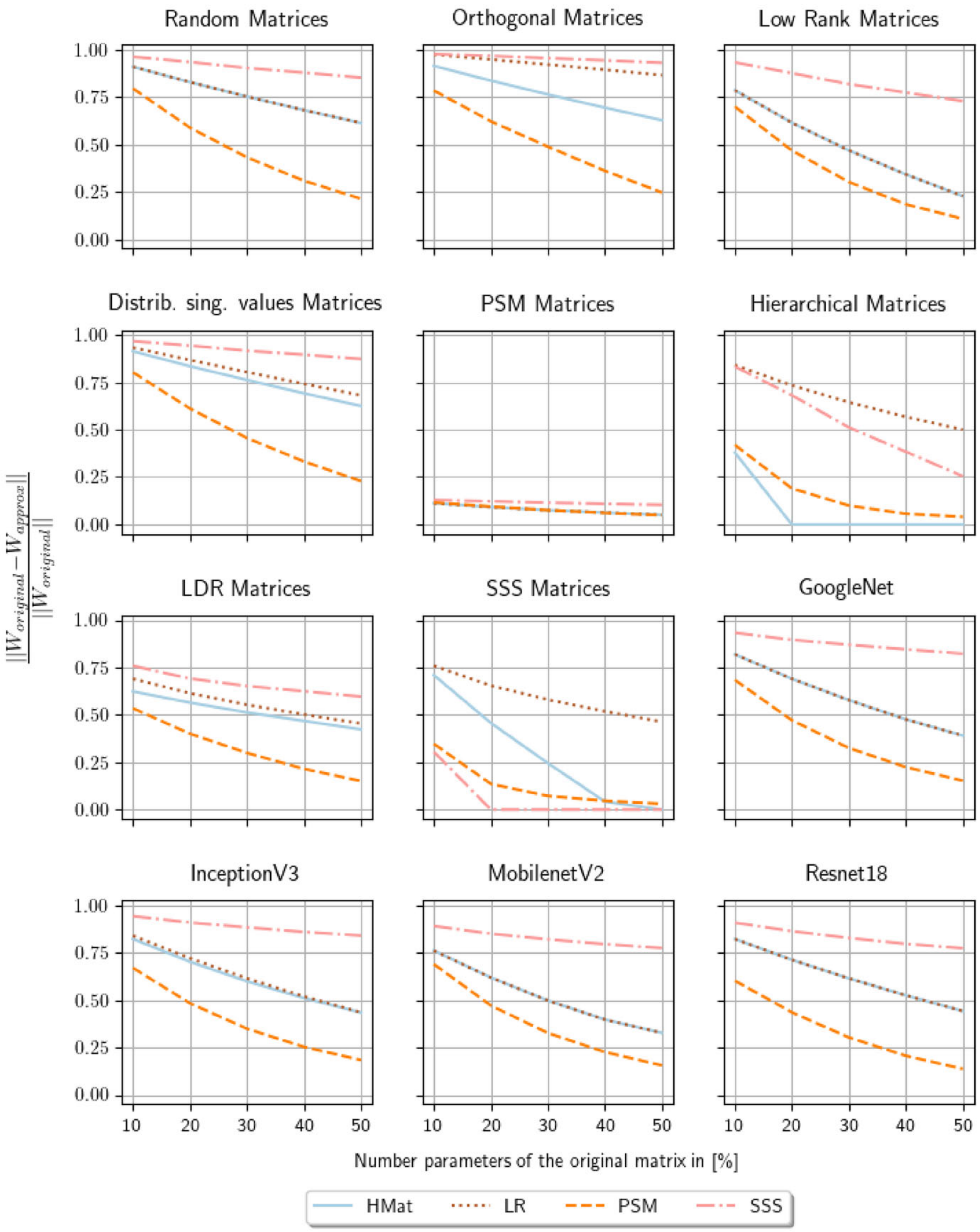

Fig. 6 Results of the approximation benchmark: We approximated several test matrices with structured matrices of different classes, namely hierarchical matrices (HMat), low-rank matrices (LR), products of sparse matrices (PSM), and sequentially semiseparable matrices (SSS). The approximation error becomes smaller if more parameters are available for approximation. If the test matrix has certain structure, we observe that the approach using the very structure performs best. In all other cases, the products of sparse matrices showed the best approximation capabilities

Table 2 Hyperparameters used during training in our fine-tuning benchmark

Hyperparameter	Value
Patience	2
Min. val. loss improvement	0.01
Loss	Cross entropy
Optimizer	Stochastic gradient descent
Number training runs η	10
Learning rate at run $i = 0, \dots, \eta - 1$	0.5^i
Batch size	150.000
Number of parameters	$\sim$ 20% of original matrix

3.3 Benchmark: Fine-Tuning

The weight matrices of neural networks are typically trained using gradient-descent (backpropagation). Considering that the backpropagation-based training led to remarkable results for neural networks in the past, we investigate the effects of training a structured weight matrix using gradient-descent. For that, we replace the last layer of pretrained PyTorch vision models by structured matrices of different classes (as explained in the previous section). Then, we fine-tune the weight matrix on the same dataset on which the original model was trained. By that, we can compare the prediction accuracy of the model before and after the fine-tuning.

We report the prediction accuracy results on the validation set, with which the models were trained originally. This validation set is *not* used during our fine-tuning. For the fine-tuning, we use a portion of the training data (randomly split before the training begins) as validation set. This validation set is used to determine when the training stops. We stop the training when the validation loss does not improve by at least 0.01 over 2 steps. For each model, there are 10 training runs based on Stochastic Gradient Descent with different learning rates. We start with learning rate $\alpha = 1$, and multiply the learning rate with 0.5 after each training run. Between training runs, we restore the model with lowest validation loss from the previous training run. All important hyperparameters can be found in Table 2.

The gradients used for training are *not* determined by deriving the prediction loss with respect to the weight matrix entries. Instead, we take the derivative of the prediction loss with respect to the parameters determining the structured weight matrix. Details about how this can be done for sequentially semiseparable weight matrices are given by Kissel et al. [51]. The gradients for the other structures can be determined analogously (in our experiments, we use the PyTorch auto differentiation tools for determining the gradients). The code used for running our experiments is available on Github.[4]

For all models, the fine-tuning was able to improve the prediction accuracy compared to the non-fine-tuned version. The accuracy improvements were smaller for models, which achieved high prediction accuracy directly after approximation. For

[4] https://github.com/MatthiasKi/structurednets.

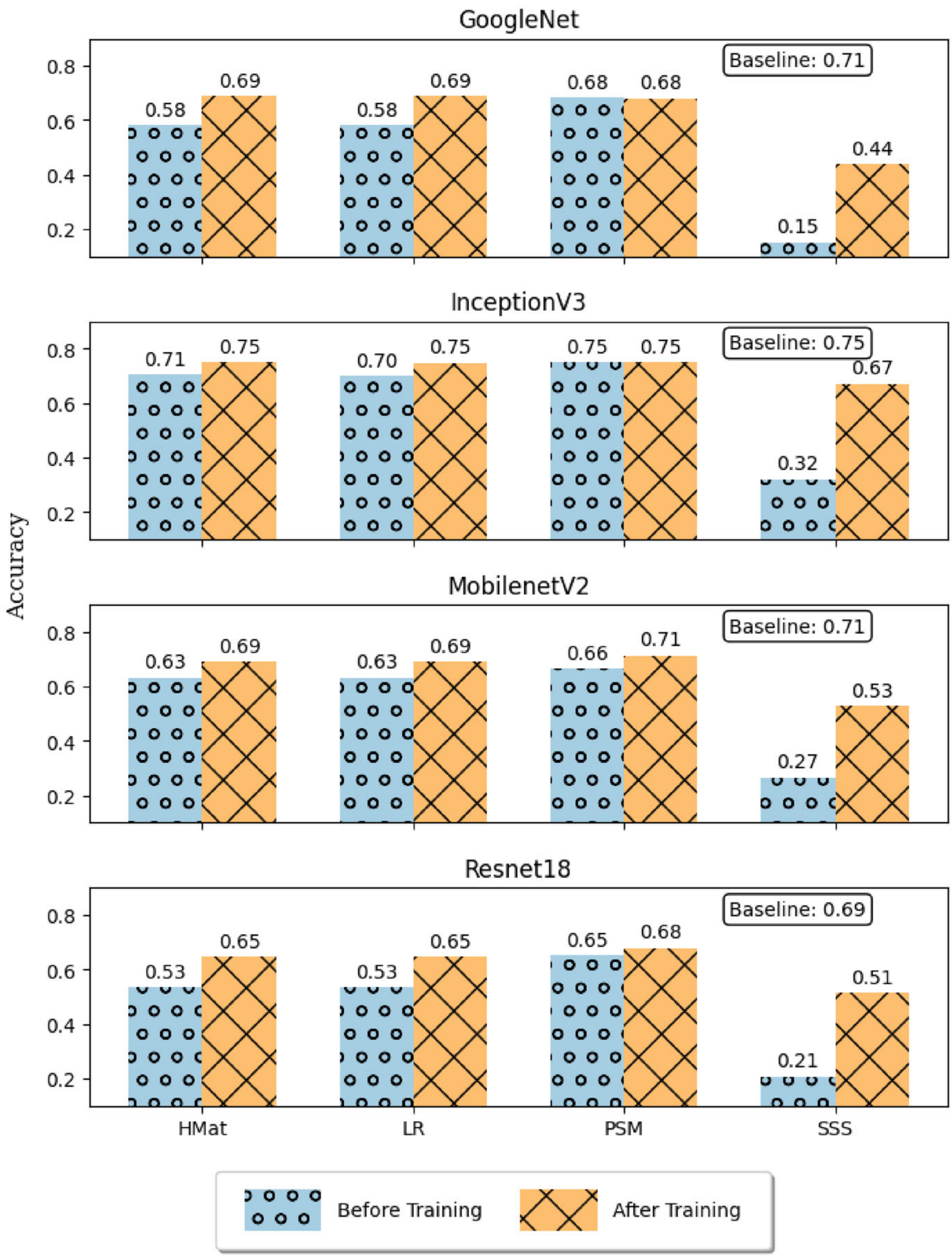

Fig. 7 Accuracy before and after fine-tuning of different PyTorch vision models. The weight matrix of the last layer is replaced with a hierarchical matrix (HMat), a low-rank matrix (LR), products of sparse matrices (PSM), or a sequentially semiseparable matrix (SSS). As expected, the fine-tuning improved the prediction accuracy in all cases. However, for the products of sparse matrices, the improvements are too small to be seen for some models in the figure (supposedly because they already showed good prediction accuracy before fine-tuning). The models with sequentially semiseparable weight matrix showed the biggest improvements. Nevertheless, their final prediction performance remains behind other structures

the products of sparse matrices, the improvements were so small, and that for some models, they are not even visible in Fig. 7.

Analogous to the results in Sect. 3.2, the models with products of sparse matrices achieved the best prediction accuracy after fine-tuning. This resulted in achieving almost the same performance as the baseline models for some of the vision models. Other structured matrices also achieved remarkable results after fine-tuning. This leads to the conclusion that for different types of structured matrices, many of the parameters can be spared while achieving almost the same results as the baseline. In this benchmark, the networks with products of sparse matrices consistently achieved the best results.

The neural networks with sequentially semiseparable weight matrix could not keep up with the performance of the other networks. They showed significant lower prediction accuracy after fine-tuning than the baseline. However, the fine-tuning led to remarkable improvements in the prediction accuracy. In all experiments, the accuracy was more than doubled after fine-tuning, which are much greater improvements than observed with other networks. This is in line with previous results, which showed that approximation of weight matrices with sequentially semiseparable matrices led to poor results [52], but by training such networks from scratch, it was possible to even increasing generalization performance [51].

4 Limitations and Discussion

The presented structures have been applied to neural networks, where they have been used for faster inference, faster training, or for network analysis. However, some questions remain unanswered to this day. In the following, we highlight two research areas in the context of neural networks with structured weight matrices for which we identified relevant unanswered questions.

Theoretical results for the use of structured matrices in neural networks are still very limited. For neural networks with weight matrices of low displacement rank, Zhao et al. [94] proved that the universal approximation theorem still holds and they gave upper bounds for the approximation error. However, proving similar results for other classes of structured matrices is still the subject of ongoing research. In particular, theoretical insights regarding approximation errors for problems with different data distributions can be helpful for selecting a suitable network. For example, they can help to decide whether a large network with structured weight matrices is preferable to a small network with standard weight matrices, depending on the problem at hand. Thus, the first research area we identified is about the question how the performance of neural networks with structured weight matrices depends on the target application. The first intuition is that the choice of a suitable structure used in the network depends very much on the application domain (as indicated by the success of CNNs in image-based domains). To our knowledge, however, this effect has not been explicitly studied yet. We consider our benchmarks as initial insights for selecting an appropriate weight matrix structure. In summary, if there is no indicator that a particular structure is suitable for the given problem, products of sparse matrices are a very good choice. These performed robustly very well in both of our benchmarks. However, we recommend to perform a hyperparameter search considering different structure classes, if the computational resources needed for the training play a minor role. This hyperparameter search might reveal another structure class that fits the problem at hand particularly well.

The second area we identified is structure-aware training. By this we mean the methodology of how structures can be introduced into the weight matrices of neural networks. In the aforementioned preliminary work, various strategies were pursued in this regard: Regularization techniques, training using backpropagation or approximation of weight matrices with structured matrices after training. But there is still limited knowledge about which method to select for a given problem. Moreover, hybrid approaches for selecting and combining the right methods could be developed. We consider the development of algorithms that find the right structure without hand-tuning and excessive expert knowledge critical to make the overall approach useful for a wide range of problems.

The aim of this paper is to give an overview over the most important structure classes and relevant structure sub-classes. However, it is of course not possible to cover all structures that have ever been studied. Therefore, we would like to mention a few structures that we did not consider.

First, we would like to mention kernel-based approaches. These are not explicit structures, which can be represented by dependencies between the matrix elements. Rather, we consider kernel-based approaches as implicit structures, since operations are spared through the kernel trick. In this context, we consider approaches that learn kernel functions from data [54], kernel-based weight matrices or layers in neural networks [14, 65].

Second, we did not address complex tensor decompositions or factorizations. For example, Yang et al. [92] showed how the adaptive fastfood transform can be used to reparameterize the matrix–vector multiplication in neural networks. Lebedev et al. [56] used a polyadic decomposition (CP decomposition) to decompose convolution kernel tensors into a sum of rank-one tensors. Moczulski et al. [63] replaced linear transformations with a product of diagonal matrices combined with the discrete cosine transform. Their *ACDC* layers can be used to replace any linear transformation in the network and is able to reduce the number of parameters from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ as well as the number of operations from $\mathcal{O}(n^2)$ to $\mathcal{O}(n\log(n))$.

5 Conclusion

In this paper, we gave an overview over the four main matrix structures and special sub-classes which they contain. We introduced each of the structure classes by showing their definition, and giving reference to research papers in which the structure is used in the domain of neural networks. Each of the presented structure classes facilitates an efficient matrix–vector multiplication algorithm. Since matrix–vector multiplications are usually the dominant factor for the computational cost of neural networks, using such structures in neural networks has the potential to reduce the required computational cost immensely, finally leading to reduced CO_2 emissions as well as reduced electricity costs.

In the two benchmarks presented in this paper, we compared the approximation capabilities of structured matrices of different classes, as well as the prediction performance of deep vision models containing structured matrices. Products of sparse matrices showed to be the most promising structure class since this structure consistently achieved good results in both benchmarks. However, choosing the right structure still depends on the problem at hand.

Our survey illustrates that the use of structured matrices in neural networks is still a fairly young research area. There are still many open questions, and we presented two research areas we consider most important in the discussion section. These are structure-aware training algorithms as well as analyzing the relationship between structured weight matrices in neural networks and the target application.

Funding Open Access funding enabled and organized by Projekt DEAL. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data availability The code and data used for running the experiments in this paper can be found on GitHub: https://github.com/MatthiasKi/structurednets. This repository uses code from the TVSCLib repository: https://github.com/MatthiasKi/tvsclib. Moreover, the experiments are based on PyTorch vision models, which are available at https://pytorch.org/.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

Research involving human participants and/or animals Not applicable.

Informed consent Not applicable.

References

1. Ailon, N., Leibovitch, O., Nair, V.: Sparse linear networks with a fixed butterfly structure: theory and practice. In: Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, vol. 161, pp. 1174–1184. PMLR (2021)
2. Ambikasaran, S.: Fast algorithms for dense numerical linear algebra and applications. PhD thesis (2013)
3. Ambikasaran, S., Darve, E.: An o (n log n) fast direct solver for partial hierarchically semi-separable matrices. J. Sci. Comput. **57**(3), 477–501 (2013)
4. Appuswamy, R., Nayak, T., Arthur, J., Esser, S., Merolla, P., Mckinstry, J., Melano, T., Flickner, M., Modha, D.: Structured convolution matrices for energy-efficient deep learning. arXiv preprint arXiv:1606.02407 (2016)
5. Beatson, R.K., Newsam, G.N.: Fast evaluation of radial basis functions: I. Comput. Math. Appl. **24**(12), 7–19 (1992)

6. Beatson, R., Greengard, L.: A short course on fast multipole methods. Wavelets Multilevel Methods Elliptic PDEs **1**, 1–37 (1997)
7. Bell, N., Garland, M.: Efficient sparse matrix-vector multiplication on cuda. Nvidia Technical Report NVR-2008-004 **2**(5) (2008)
8. Blalock, D., Ortiz, J.J.G., Frankle, J., Guttag, J.: What is the state of neural network pruning? arXiv preprint arXiv:2003.03033 (2020)
9. Börm, S., Grasedyck, L., Hackbusch, W.: Hierarchical matrices. Lect. Notes **21**, 2003 (2003)
10. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory, vol. 315. Springer Science & Business Media, New York (2013)
11. Chandrasekaran, S., Ming, G., Pals, T.: A fast ULV decomposition solver for hierarchically semiseparable representations. SIAM J. Matrix Anal. Appl. **28**(3), 603–622 (2006)
12. Chen, C., Reiz, S., Yu, C.D., Bungartz, H.-J., Biros, G.: Fast approximation of the Gauss–Newton hessian matrix for the multilayer perceptron. SIAM J. Matrix Anal. Appl. **42**(1), 165–184 (2021)
13. Cheng, Y., Felix, X.Y., Feris, R.S., Kumar, S., Choudhary, A., Chang, S.-F.: Fast neural networks with circulant projections. arXiv preprint arXiv:1502.03436 (2015)
14. Cho, Y.: Kernel methods for deep learning. PhD thesis, UC San Diego (2012)
15. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**(90), 297–301 (1965)
16. Dao, T., Gu, A., Eichhorn, M., Rudra, A., Ré, C.: Learning fast algorithms for linear transforms using butterfly factorizations. In: International Conference on Machine Learning, pp. 1517–1527. PMLR (2019)
17. Dao, T., Sohoni, N., Gu, A., Eichhorn, M., Blonder, A., Leszczynski, M., Rudra, A., Ré, C.: Kaleidoscope: An efficient, learnable representation for all structured linear maps. In: International Conference on Learning Representations (2020)
18. Darve, E.: The fast multipole method: numerical implementation. J. Comput. Phys. **160**(1), 195–240 (2000)
19. De Sa, C., Cu, A., Puttagunta, R., Ré, C., Rudra, A.: A two-pronged progress in structured dense matrix vector multiplication. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1060–1079. SIAM (2018)
20. de Sousa, C., Hemerly, E.M., Galvão, R.K.H.: Adaptive control for mobile robot using wavelet networks. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **32**(4), 493–504 (2002)
21. Dettmers, T., Zettlemoyer, L.: Sparse networks from scratch: faster training without losing performance. arXiv preprint arXiv:1907.04840 (2019)
22. Dewilde, P., Van der Veen, A.-J.: Time-Varying Systems and Computations. Springer Science & Business Media, New York (1998)
23. Eidelman, Y., Gohberg, I.: Inversion formulas and linear complexity algorithm for diagonal plus semiseparable matrices. Comput. Math. Appl. **33**(4), 69–79 (1997)
24. Eidelman, Y., Gohberg, I.: On a new class of structured matrices. Integr. Equ. Oper. Theory **34**(3), 293–324 (1999)
25. Ejbali, R., Zaied, M.: A dyadic multi-resolution deep convolutional neural wavelet network for image classification. Multimed. Tools Appl. **77**(5), 6149–6163 (2018)
26. ElAdel, A., Ejbali, R., Zaied, M., Amar, C.B.: Dyadic multi-resolution analysis-based deep learning for Arabic handwritten character classification. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 807–812. IEEE (2015)
27. Fan, Y., Lin, L., Ying, L., Zepeda-Núnez, L.: A multiscale neural network based on hierarchical matrices. Multiscale Model. Simul. **17**(4), 1189–1213 (2019)
28. Flegar, G., Anzt, H.: Overcoming load imbalance for irregular sparse matrices. In: Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms, pp. 1–8 (2017)
29. Flegar, G., Quintana-Ortí, E.S.: Balanced CSR sparse matrix-vector product on graphics processors. In: European Conference on Parallel Processing, pp. 697–709. Springer (2017)
30. Fong, W., Darve, E.: The black-box fast multipole method. J. Comput. Phys. **228**(23), 8712–8725 (2009)
31. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2018)
32. Galvão, R.K.H., Yoneyama, T.: A competitive wavelet network for signal clustering. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **34**(2), 1282–1288 (2004)

33. Gantmakher, F., Krein, M.: Sur les matrices completement non négatives et oscillatoires. Compos. Math. **4**, 445–476 (1937)
34. Gaudet, C.J., Maida, A.S.: Deep quaternion networks. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2018)
35. Geus, R., Röllin, S.: Towards a fast parallel sparse symmetric matrix-vector multiplication. Parallel Comput. **27**(7), 883–896 (2001)
36. Giffon, L., Ayache, S., Kadri, H., Artières, T., Sicre, R.: Psm-nets: compressing neural networks with product of sparse matrices. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2021)
37. Goumas, G., Kourtis, K., Anastopoulos, N., Karakasis, V., Koziris, N.: Understanding the performance of sparse matrix-vector multiplication. In: 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), pp. 283–292. IEEE (2008)
38. Grasedyck, L.: Hierarchical singular value decomposition of tensors. SIAM J. Matrix Anal. Appl. **31**(4), 2029–2054 (2010)
39. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. J. Comput. Phys. **73**(2), 325–348 (1987)
40. Greengard, L., Rokhlin, V.: A new version of the fast multipole method for the Laplace equation in three dimensions. Acta Numer. **6**, 229–269 (1997)
41. Hackbusch, W.: Hierarchical Matrices: Algorithms and Analysis, vol. 49. Springer, New York (2015)
42. Hackbusch, W., Börm, S.: Data-sparse approximation by adaptive 2-matrices. Computing **69**(1), 1–35 (2002)
43. Hackbusch, W., Grasedyck, L., Börm, S.: An introduction to hierarchical matrices. Math. Bohem. **2**, 101–111 (2002)
44. Hassibi, B., Stork, D.G.: Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. Morgan Kaufmann, Burlington (1993)
45. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
46. Hill, P., Jain, A., Hill, M., Zamirai, B., Hsu, C.-H., Laurenzano, M.A., Mahlke, S., Tang, L., Mars, J.: DeftNN: addressing bottlenecks for DNN execution on GPUS via synapse vector elimination and near-compute data fission. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 786–799 (2017)
47. Im, E.-J.: Optimizing the performance of sparse matrix-vector multiplication. PhD thesis (2000)
48. Ithapu, V.K.: Decoding the deep: Exploring class hierarchies of deep representations using multiresolution matrix factorization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 45–54 (2017)
49. Jemai, O., Zaied, M., Amar, C.B., Alimi, M.A.: Fast learning algorithm of wavelet network based on fast wavelet transform. Int. J. Pattern Recognit. Artif. Intell. **25**(08), 1297–1319 (2011)
50. Kissel, M., Diepold, K.: Deep convolutional neural networks with sequentially semiseparable weight matrices. ESANN 2022 Proceedings (2022)
51. Kissel, M., Gottwald, M., Gjeroska, B., Paukner, P., Diepold, K.: Backpropagation through states: training neural networks with sequentially semiseparable weight matrices. In: Proceedings of the 21st EPIA Conference on Artificial Intelligence (2022)
52. Kissel, M., Gronauer, S., Korte, M., Sacchetto, L., Diepold, K.: Exploiting structures in weight matrices for efficient real-time drone control with neural networks. In: Proceedings of the 21st EPIA Conference on Artificial Intelligence (2022)
53. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. **25**, 1097–1105 (2012)
54. Le, L., Hao, J., Xie, Y., Priestley, J.: Deep kernel: learning kernel function from data using deep neural network. In: Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, pp. 1–7 (2016)
55. Le Magoarou, L., Gribonval, R.: Flexible multilayer sparse approximations of matrices and applications. IEEE J. Select. Top. Signal Process. **10**(4), 688–700 (2016)
56. Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In: 3rd International Conference on Learning Representations, ICLR 2015-Conference Track Proceedings (2015)
57. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605 (1990)

58. Li, Y., Cheng, X., Jianfeng, L.: Butterfly-net: optimal function representation based on convolutional neural networks. Commun. Comput. Phys. **28**(5), 1838–1885 (2020)
59. Li, Y., Yang, H., Martin, E.R., Ho, K.L., Ying, L.: Butterfly factorization. Multiscale Model. Simul. **13**(2), 714–732 (2015)
60. Liao, S., Yuan, B.: Circconv: a structured convolution with low complexity. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 4287–4294 (2019)
61. Liu, Y., Jiao, S., Lim, L.-H.: Lu decomposition and Toeplitz decomposition of a neural network. arXiv preprint arXiv:2211.13935 (2022)
62. Mallat, S.G.: A theory for multiresolution signal decomposition: the wavelet representation. IEEE Trans. Pattern Anal. Mach. Intell. **11**(7), 674–693 (1989)
63. Moczulski, M., Denil, M., Appleyard, J., de Freitas, N., Wang, Z., Zoghi, M., Hutter, F., Matheson, D., de Freitas, N., Reed, S., et al.: Acdc: a structured efficient linear layer. In: International Conference on Learning Representations (ICLR), vol. 55, pp. 1005–1014. Universities of Harvard, Oxford, and Google DeepMind
64. Monakov, A., Lokhmotov, A., Avetisyan, A.: Automatically tuning sparse matrix-vector multiplication for GPU architectures. In: International Conference on High-Performance Embedded Architectures and Compilers, pp. 111–125. Springer (2010)
65. Muller, L., Martel, J., Indiveri, G.: Kernelized synaptic weight matrices. In: International Conference on Machine Learning, pp. 3654–3663. PMLR (2018)
66. O'Shea, K., Nash, R.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 (2015)
67. Pan, V.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Springer Science & Business Media, New York (2001)
68. Parcollet, T., Morchid, M., Linarès, G.: Quaternion convolutional neural networks for heterogeneous image processing. In: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 8514–8518. IEEE (2019)
69. Parker, D.S.: Random butterfly transformations with applications in computational linear algebra (1995)
70. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32, pp. 8024–8035. Curran Associates, Inc. (2019)
71. Pati, Y.C., Krishnaprasad, P.S.: Analysis and synthesis of feedforward neural networks using discrete affine wavelet transformations. IEEE Trans. Neural Netw. **4**(1), 73–85 (1993)
72. Pichel, J.C., Heras, D.B., Cabaleiro, J.C., Rivera, F.F.: Improving the locality of the sparse matrix-vector product on shared memory multiprocessors. In: 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2004. Proceedings, pp. 66–71. IEEE (2004)
73. Pinar, A., Heath, M.T.: Improving performance of sparse matrix-vector multiplication. In: SC'99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing, p. 30. IEEE (1999)
74. Postalcioglu, S., Becerikli, Y.: Wavelet networks for nonlinear system modeling. Neural Comput. Appl. **16**(4), 433–441 (2007)
75. Rózsa, P., Bevilacqua, R., Romani, F., Favati, P.: On band matrices and their inverses. Linear Algebra Appl. **150**, 287–295 (1991)
76. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia (2003)
77. Shanks, J.L.: Computation of the fast Walsh–Fourier transform. IEEE Trans. Comput. **100**(5), 457–459 (1969)
78. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
79. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
80. Sindhwani, V., Sainath, T.N., Kumar, S.: Structured transforms for small-footprint deep learning. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 2, pp. 3088–3096 (2015)

81. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in nlp. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 3645–3650 (2019)
82. Sze, V., Chen, Y.-H., Yang, T.-J., Emer, J.S.: Efficient processing of deep neural networks: a tutorial and survey. Proc. IEEE **105**(12), 2295–2329 (2017)
83. Temam, O., Jalby, W.: Characterizing the behavior of sparse algorithms on caches. PhD thesis, INRIA (1992)
84. Thomas, A.T., Albert, G., Dao, T., Rudra, A., Ré, C.: Learning compressed transforms with low displacement rank. Adv. Neural Inf. Process. Syst. **2018**, 9052 (2018)
85. Toledo, S.: Improving the memory-system performance of sparse-matrix vector multiplication. IBM J. Res. Dev. **41**(6), 711–725 (1997)
86. Vandebril, R., Van Barel, M., Golub, G., Mastronardi, N.: A bibliography on semiseparable matrices. Calcolo **42**(3), 249–270 (2005)
87. Vandebril, R., Van Barel, M., Mastronardi, N.: Matrix Computations and Semiseparable Matrices: Linear Systems, vol. 1. JHU Press, Baltimore (2007)
88. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, pp. 2082–2090 (2016)
89. Wu, B., Wang, D., Zhao, G., Deng, L., Li, G.: Hybrid tensor decomposition in neural network compression. Neural Netw. **132**, 309–320 (2020)
90. Xie, D., Xiong, J., Pu, S.: All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6176–6185 (2017)
91. Xu, Z., Li, Y., Cheng, X.: Butterfly-net2: simplified butterfly-net and Fourier transform initialization. In: Mathematical and Scientific Machine Learning, pp. 431–450. PMLR (2020)
92. Yang, Z., Moczulski, M., Denil, M., De Freitas, N., Smola, A., Song, L., Wang, Z.: Deep fried convnets. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1476–1483 (2015)
93. Zhang, Q., Benveniste, A.: Wavelet networks. IEEE Trans. Neural Netw. **3**(6), 889–898 (1992)
94. Zhao, L., Liao, S., Wang, Y., Li, Z., Tang, J., Yuan, B.: Theoretical properties for neural networks with weight matrices of low displacement rank. In: International Conference on Machine Learning, pp. 4082–4090. PMLR (2017)
95. Zhu, X., Xu, Y., Xu, H., Chen, C.: Quaternion convolutional neural networks. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 631–647 (2018)

5.2. Backpropagation Through States: Training Neural Networks with Sequentially Semiseparable Weight Matrices

Authors: Matthias Kissel, Martin Gottwald, Biljana Gjeroska, Philipp Paukner, and Klaus Diepold
Journal: Progress in Artificial Intelligence (Proceedings of the 21st EPIA Conference on Artificial Intelligence)
Publisher, publication year: Springer, 2022
Core publication: Yes
Pages: 476–487
Bibliography Entry: [51]
Comment: Reproduced with permission from Springer Nature

Summary

During the training of NNs with SSS weight matrices, it must be ensured that the SSS structure does not vanish. For that, we introduce a training algorithm called *Backpropagation through states* in this paper (recapitulated in Section 6.4 of this thesis). Using this algorithm, NNs with SSS weight matrices can be trained end-to-end, whereas it is guaranteed that the weight matrix remains structured throughout the training. In order to benchmark NNs trained with the backpropagation through states algorithm, we analyze the prediction performance of trained NNs on several standard benchmark problems. I use the results of these benchmarks in Section 7.1 as evidence that NNs with SSS weight matrices can outperform standard NNs in terms of prediction accuracy. Moreover, we show in this paper that depending on the hardware, using SSS weight matrices can lead to a reduction in computation time for computing the matrix-vector product. In particular, using SSS weight matrices on the microcontroller introduced in the motivational example in Section 1.2 of this thesis can result in faster computations.

Own Contributions

- Literature review of previous work in the field and comparable approaches
- Mathematical formulation and analysis of the presented algorithm in cooperation with Martin Gottwald
- Design, implementation and execution of the experiments listed in the paper based on code examples by Martin Gottwald and Biljana Gjeroska
- Analysis of the time needed to use the presented method in collaboration with Mathias Korte

Backpropagation Through States: Training Neural Networks with Sequentially Semiseparable Weight Matrices

Matthias Kissel(✉), Martin Gottwald, Biljana Gjeroska, Philipp Paukner, and Klaus Diepold

Technical University of Munich, Arcisstr. 21, 80333 Munich, Germany
matthias.kissel@tum.de
https://www.tum.de/en/

Abstract. Matrix-Vector multiplications usually represent the dominant part of computational operations needed to propagate information through a neural network. This number of operations can be reduced if the weight matrices are structured. In this paper, we introduce a training algorithm for neural networks with sequentially semiseparable weight matrices based on the backpropagation algorithm. By exploiting the structures in the weight matrices, the computational complexity for computing the matrix-vector product can be reduced to the subquadratic domain. We show that this can lead to computing time reductions on a microcontroller. Furthermore, we analyze the generalization capabilities of neural networks with sequentially semiseparable matrices. Our experiments show that neural networks with structured weight matrices can outperform standard feed-forward neural networks in terms of test prediction accuracy for several real-world datasets.

Keywords: Structured matrices · Neural networks · Efficient inference

1 Introduction

In recent years, the trend for neural networks has been towards larger and deeper networks [8,20]. Together with the size of the networks, the demand for computing resources also increased. For example, the number of operations needed to propagate information through the neural network can significantly increase with the network width. This limits the usability of neural networks for many applications, especially for real-time applications or on mobile platforms.

The major computational costs for propagating information through a neural network are typically attributed to matrix vector products. At each layer, the inputs are multiplied with the weight matrix of the layer, which amounts to $\mathcal{O}(nm)$ operations (for a weight matrix $W \in \mathcal{R}^{m \times n}$). These computational costs can be reduced if the weight matrix possesses a specific structure. This is due

G. Marreiros et al. (Eds.): EPIA 2022, LNAI 13566, pp. 476–487, 2022.
https://doi.org/10.1007/978-3-031-16474-3_39

to the fact that for some matrix structures, there are efficient algorithms for multiplying the matrix with a vector with subquadratic order of operations.

Therefore, if the weight matrices of a neural network are structured, the number of operations for propagating information through the network can be reduced significantly. It has been observed that weights of a neural network tend to be structured after training [3]. Besides observing the structure *after* training, one can also enforce structure in the weight matrices *during* training. This has been shown for example by Sindhwani et al. [12] or Thomas et al. [14] for matrices with low displacement rank.

In this paper, we focus on Sequentially Semiseparable Matrices, which are related to linear time-varying system theory [4]. Our contribution is two-fold. First, we introduce a training algorithm for neural networks with sequentially semiseparable weight matrices. Our algorithm ensures that the weight matrices remain structured while optimizing the training error. Second, we compare the generalization performance of structured neural networks with standard feed-forward neural networks on four real-world datasets.

The paper is organized as follows. We first give an overview over approaches of using structured matrices in neural networks and work connecting semiseparable matrices with neural networks in literature. In the subsequent section, we introduce neural networks with sequentially semiseparable weight matrices. Afterwards, we present our training algorithm *Backpropagation through states*. The results of our experiments are shown and discussed in Sect. 5. Finally, we summarize our findings and draw a conclusion.

2 Literature Review

Several approaches for finding structure in trained weight matrices, or imposing structure constraints during training have been proposed recently. Here, most often matrices of low displacement rank have been used in neural networks. For example, Sindhwani et al. [12] proposed to train neural networks with toeplitz-like weight matrices and Thomas et al. [14] introduced a class of low displacement rank matrices, which can be trained end-to-end including the operator matrices. Zhao et al. [22] proved some theoretical properties for neural networks with weight matrices of low displacement rank.

Another structure, which has been applied to neural networks, are hierarchical matrices [5]. Connecting Hierarchical matrices with semiseparable matrices results in hierarchically semiseparable matrices [2,18].

Finding the right structure for a given problem is difficult, especially since the right structure depends on the problem at hand. Therefore, we regard the previously mentioned approaches not as competitors, but as complementary approaches. For a specific problem, one of the structures from literature might work very good, and for another problem the structure analyzed in this paper might be better. Hence, we think it is crucially important to have several structure-aware training or approximation methods for neural networks, in order to find the best approach for a given problem.

In this paper, we are focusing on sequentially semiseparable matrices. The concept of this structure dates back until 1937 [6,17]. Sequentially semiseparable matrices are closely related to the theory of time varying systems [4], since this structure appears when describing a time varying system. Other definitions of semiseparability have been introduced by Vandebril et al. [18] - for example for quasiseparable matrices.

To the best of our knowledge, (sequentially) semiseparable matrices have not been applied as weight matrices in neural networks yet. Related work, which used semiseparable matrices in the domain of neural networks focused on finding suitable neural network architectures for time-varying system applications. For example, the aim of State-Space Neural Networks [16,21] is to introduce non-linearity into the state space representation of time-varying systems. Another example are Time-Varying Neural Networks (TV-NN) [15], in which the weights of the network change over time in order to adapt to non-stationary input signals. Our method differs from such approaches in that we do not intend to design application specific network architectures. Instead, in our approach we constrain the weight matrices in neural networks to have sequentially semiseparable structure. Our approach refers generically to neural networks, and explicitly not to a specific target application.

3 Neural Networks with Sequentially Semiseparable Weight Matrices

We define neural networks as function $G(u)$

$$\hat{y} = G(u), \tag{1}$$

where u are the inputs to the network and $\hat{y}$ the outputs respectively. G is a composition of layer mappings

$$G(u) = (\mathcal{L}_r \circ \cdots \circ \mathcal{L}_1)(u), \tag{2}$$

where the neural network consists of r layers and $\mathcal{L}_i$ is the mapping of the i^{th} layer. In this paper, we focus on structures in densely connected feed-forward neural networks. For these layers, the mappings are of the form

$$\mathcal{L}_i(u) = \sigma(W_i u + \theta_i), \tag{3}$$

where W_i is a weight matrix and θ_i the biases of the respective layer. σ is the activation function of the layer, which is applied element-wise to its inputs.

We are interested in the special case that the weight matrices W_i are structured. In particular, we want them to be sequentially semiseparable, which allows us to use results from time-varying systems theory [4] to increase the efficiency of information propagation. Sequentially semiseparable matrices can be expressed as

$$\begin{aligned} W_i = D &+ C(I - ZA)^{-1}ZB \\ &+ G(I - Z^T E)^{-1} Z^T F. \end{aligned} \tag{4}$$

Here, I is the identity matrix and Z is a down-shift matrix

$$Z = \begin{pmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & 0 \\ 0 & & 1 & 0 \end{pmatrix}. \quad (5)$$

A, B, C, D, E, F and G are block-diagonal matrices, each comprising of k matrices

$$A = diag([A_1, \dots, A_k]) \quad (6)$$

(B, C, D, E, F and G matrices respectively). In the context of time-varying system theory, the matrices $A, \dots, G$ define the behavior of a time-varying system. For example, A maps the previous state of the system to the next state, and B maps the previous state to the current output. Note that the dimensions of the A_k, B_k, C_k, D_k, E_k, F_k and G_k matrices are not constant. In general we have $dim(A_i) \neq dim(A_j)$ for $i \neq j$ (B_k, C_k, D_k, E_k, F_k and G_k matrices respectively). This reflects the fact that the state, input and output dimension can change for different k.

In order to apply the results from time varying system theory to our matrix vector products $W_i u$, the input vector u as well as the output vector $\hat{y}$ must be partitioned into p segments

$$u = \begin{pmatrix} u_1 \dots u_p \end{pmatrix}^T \quad (7)$$

($\hat{y}$ respectively). Note that both, u and $\hat{y}$, must be partitioned into the same amount of segments. However, the segments can be of different size, which means that in general

$$dim(u_j) \neq dim(\hat{y}_j) \quad for \quad j = 1 \dots p. \quad (8)$$

Finding a good partitioning depends on the problem at hand. In our experiments, we set $dim(u_j) = dim(\hat{y}_j) = 1$ for $j = 1 \dots p$. This results in $p = n$ for square weight matrices $W \in \mathcal{R}^{n \times n}$.

Exploiting the structure of our weight matrices, the product between W_i and an arbitrary input vector u can be performed in the state-space representation. The corresponding state equations are

$$x_{k+1} = A_k x_k + B_k u_k \quad (9)$$

$$\hat{x}_k = E_k \hat{x}_{k+1} + F_k u_k \quad (10)$$

$$\hat{y}_k^{(1)} = C_k x_k + D_k u_k \quad (11)$$

$$\hat{y}_k^{(2)} = G_k \hat{x}_{k+1} \quad (12)$$

$$\hat{y}_k = \hat{y}_k^{(1)} + \hat{y}_k^{(2)}, \quad (13)$$

where x_{k+1} is the state of the causal part of the matrix, and $\hat{x}_k$ the state of the anti-causal part respectively. The computational graph of these operations is illustrated in Fig. 1.

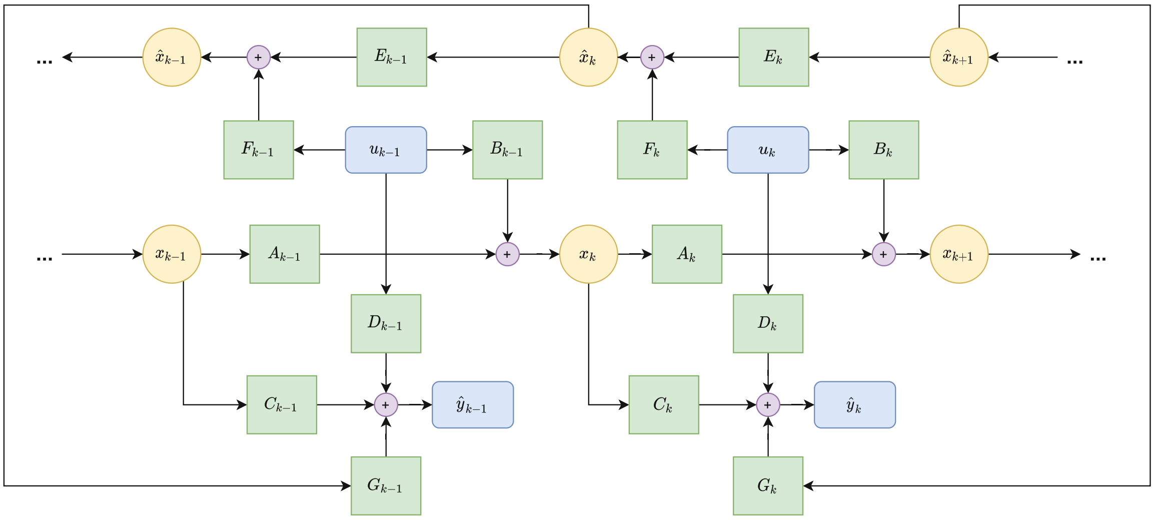

Fig. 1. Computational graph for computing the matrix-vector product in state space. This illustrates the operations described by Eqs. 9–13. In terms of time varying systems, x_k and $\hat{x}_k$ describe the state of the system, u_k are inputs and $\hat{y}_k$ are outputs.

We usually have

$$\max_k\{dim(x_k)\} = \max_k\{dim(\hat{x}_k)\} = d. \tag{14}$$

Moreover, we assume that

$$\max_k\{dim(u_k)\} < d \tag{15}$$

and

$$\max_k\{dim(\hat{y}_k)\} < d. \tag{16}$$

In this scenario, the computational complexity for computing the matrix-vector product for a square weight matrix $W \in \mathcal{R}^{n\times n}$ reduces from $\mathcal{O}(n^2)$ to $\mathcal{O}(pd^2)$. We run experiments with values of d in the range $d = 1, \dots 5$.

4 Backpropagation Through States

We consider neural networks as defined in Eq. 2 with at least one weight matrix of the form given in Eq. 4. If such a network would be trained with the standard backpropagation algorithm, the structure would most probably vanish during training. That is, after updating a structured weight matrix W according to the gradient taken with respect to the entries of the matrix $W_{e,l}$, the matrix is in general not sequentially semiseparable anymore. To solve this, we propose our training algorithm *Backpropagation through States*, which ensures that the matrix stays sequentially semiseparable after updating the weights. We introduce the necessary steps for a given linear layer with sequentially semiseparable weight matrix W. These steps can be combined with the standard backpropagation steps

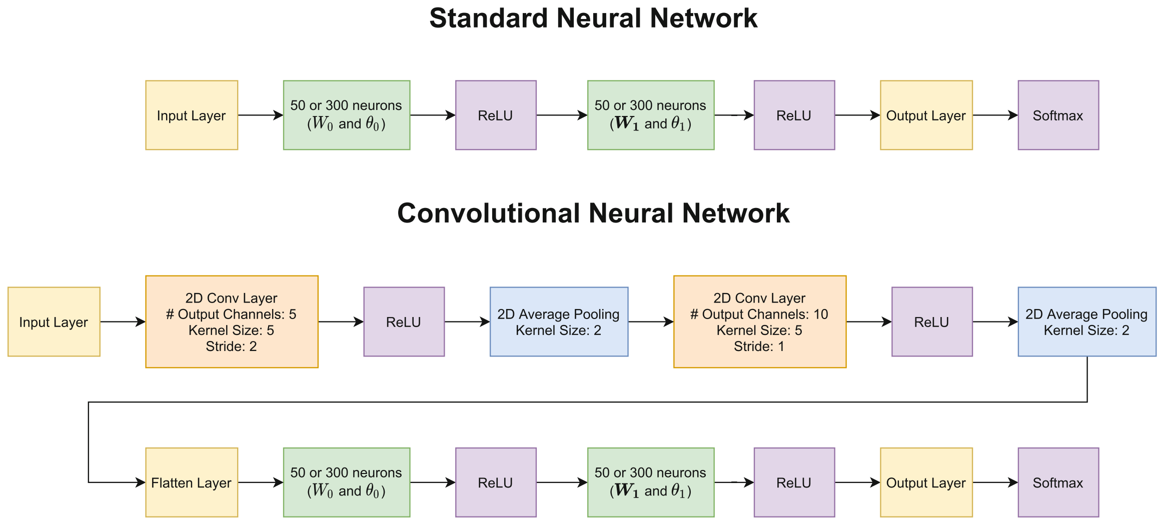

Fig. 2. Neural network architectures used in our experiments. The CNN architecture is used for image-based datasets, and the standard architecture in all others. The classifier part consisting of two hidden layers stays the same for our standard networks and convolutional networks. The feature extractor part used for image-based datasets consists of two convolutional layers followed by pooling layers. In our experiments, we focus on the weight matrix $\boldsymbol{W_1}$. We compare the generalization performance for W_1 being a sequentially semiseparable matrix, a rank 1 matrix or a standard weight matrix.

for the rest of the network, which might as well contain non-fully connected parts like convolutional layers.

The key idea of *Backpropagation through states* is to derive the training error with respect to the entries in the A_k, B_k, C_k, D_k, E_k, F_k and G_k matrices instead of the entries in W. We illustrate the approach in the following exemplary for the setting $dim(u_k) = dim(\hat{y}_k) = 1$ for $k = 1 \ldots p$.

Figure 1 depicts the data flow in the state space model for computing the outputs $\hat{y}$. C_k, D_k and G_k do not influence the state of the system. Therefore, these matrices contribute only to a single output segment

$$\frac{\delta L(y,\hat{y})}{\delta C_k} = \frac{\delta L(y,\hat{y})}{\delta \hat{y}_k} x_k^T, \tag{17}$$

$$\frac{\delta L(y,\hat{y})}{\delta D_k} = \frac{\delta L(y,\hat{y})}{\delta \hat{y}_k} u_k, \tag{18}$$

$$\frac{\delta L(y,\hat{y})}{\delta G_k} = \frac{\delta L(y,\hat{y})}{\delta \hat{y}_k} \hat{x}_{k+1}^T, \tag{19}$$

where $L(y, \hat{y})$ refers to the loss based on the desired output y and the predicted output $\hat{y}$.

In contrast, A_k, B_k, E_k and F_k change the state of the system. By that, they can contribute to past or future outputs (depending if the matrices belong to the causal or anticausal part). During backpropagation, the influence of these matrices for other output segments must also be considered, which results in

$$\frac{\delta L(y,\hat{y})}{\delta A_k} = \sum_{s=k+1}^{p} \frac{\delta L(y,\hat{y})}{\delta \hat{y}_s} \frac{\delta(\tilde{C}(s,k)A_k x_k)}{\delta A_k} \tag{20}$$

$$= \sum_{s=k+1}^{p} \frac{\delta L(y,\hat{y})}{\delta \hat{y}_s} \left(x_k^T \otimes \tilde{C}(s,k)\right) \tag{21}$$

and

$$\frac{\delta L(y,\hat{y})}{\delta B_k} = \sum_{s=k+1}^{p} \frac{\delta L(y,\hat{y})}{\delta \hat{y}_s} \frac{\delta(\tilde{C}(s,k)B_k u_k)}{\delta B_k} \tag{22}$$

$$= \sum_{s=k+1}^{p} \frac{\delta L(y,\hat{y})}{\delta \hat{y}_s} \left(u_k^T \otimes \tilde{C}(s,k)\right), \tag{23}$$

where $\otimes$ denotes the Kronecker product and

$$\tilde{C}(s,k) = C_s \prod_{k+1}^{f=s-1} A_f. \tag{24}$$

The gradients for the anticausal part (E_k and F_k) can be computed analogously.

In order to compute the gradients for A_k, B_k, E_k and F_k, the error gets propagated through the states, which gives our algorithm its name. In our experiments, we used auto-differentiation provided by pytorch[1] to compute the gradients. Our code can be found on GitHub[2].

Empirically, we noticed that initializing the $A_k, B_k, C_k, D_k, E_k, F_k$ and G_k matrices randomly often leads to numerical instability. The resulting weight matrix might have a very big condition number, which led to problems during inference as well as during training (vanishing and exploding gradients). In order to overcome this problem, we propose to initialize the weight matrix W glorot-uniform randomly. The required parameter matrices are then obtained by performing balanced model reduction [4,10] on the randomly initialized weight matrix. By that, the training procedure becomes more stable, while still allowing for indirectly randomly initialized parameter matrices. Note that the weight matrix reconstructed from the parameter matrices obtained by balanced model reduction in general differs from the original glorot-uniform initialized weight matrix.

5 Experiments and Discussion

In our experiments we investigate the prediction accuracy of neural networks with a sequentially semiseparable weight matrix compared to standard neural networks. The architectures of the networks used in our experiments are depicted

[1] https://pytorch.org/.
[2] https://github.com/MatthiasKi/statespace_learning.

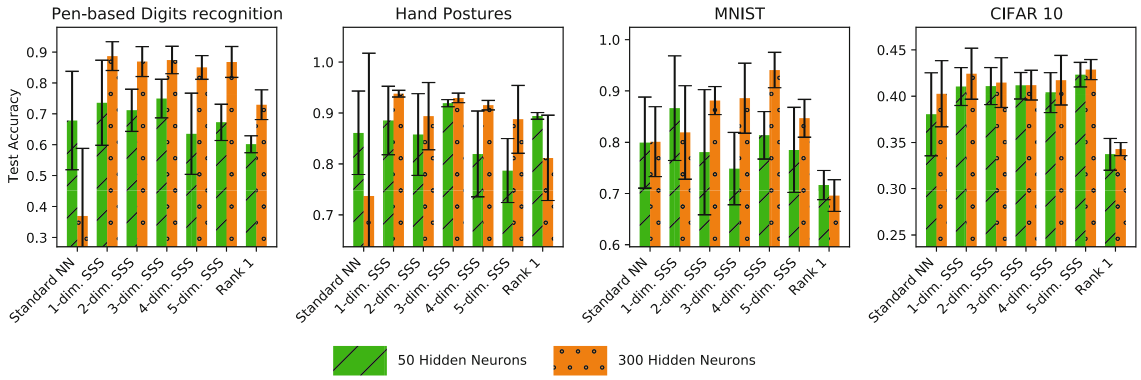

Fig. 3. Generalization performance of standard neural networks, neural networks with one sequentially semiseparable weight matrix, and neural networks with one rank 1 weight matrix on different datasets. Neural networks with semiseparable weight matrices consistently achieve a better test accuracy. This effect is especially visible for models with 300 neurons. In contrast, the rank 1 weight matrix approach could not improve the test prediction accuracy on all datasets.

in Fig. 2. In our comparison, we focus on the weight matrix between the first and the second densely connected hidden layer, corresponding to W_1 in the Figure. We investigate the effect of constraining W_1 to be sequentially semiseparable compared to a non-restricted weight matrix or a weight matrix of rank 1. The number of neurons in the hidden layers was set to 300 so that the parameters in the weight matrix under study account for approximately 90% of the total number of parameters in the neural network (assuming that the parameters in the convolutional layers are shared). In order to compare the results with the scenario where the weight matrix does not represent the dominant part of the parameters, we also perform all experiments with 50 neurons in the hidden layers.

All experiments are repeated 5 times to account for random initialization of the weight matrices and stochastic effects. We chose 5 repetitions, because this was still reasonable considering the computation time. Moreover, we assume that with this number of repetitions the effect of the random initialization can be estimated well. We report the mean and standard deviation of the test prediction accuracy over those runs. Each model has been trained for 200 epochs, followed by training until convergence on a validation set, which comprises 15% of the training data (randomly split at the beginning of the training). Thus, the number of epochs is determined data-driven, and is not set in advance as a hyperparameter. The most important hyperparameters are listed in Table 2.

We train all models on four real-world datasets (see Table 1), inter alia obtained from the UCI machine learning repository[3]:

– The Pen-Based Digit Recognition dataset [1], which contains resampled coordinates of individuals drawing digits.
– The Motion Capture Hand Postures dataset [7], for which the aim is to predict the correct hand posture given the coordinates of 11 markers. There are many

[3] https://archive.ics.uci.edu/.

Table 1. Characteristics of the datasets used in the experiments.

Name	Inp. Dim.	Classes	# Train	# Test
Pendigits	16	10	7494	3,498
Hand postures	36	5	62,476	15,619
MNIST	(28, 28)	10	60,000	10,000
CIFAR10	(3, 32, 32)	10	50,000	10,000

Table 2. Overview over hyperparameters used in our experiments.

Hyperparameter	Value
Repetitions per experiment	5
$dim(u_k), dim(a_k) \quad \forall k$	1
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, lr = 1e-3$)
Training loss function	Cross entropy
Validation set share	15%
Convergence patience	20

missing values in the dataset (which we mask with zeros) and the marker positions have been permuted between different recordings. At each iteration, we randomly split the samples in the datapoint into training (80% of the data) and test (20% of the data) set.
- The MNIST digits classification dataset, which contains grayscale images of handwritten digits.
- The CIFAR10 images classification dataset, which comprises color images from 10 different categories.

We use the proposed split of the data into training and testing data if not stated otherwise.

Our experiments show that the generalization performance can be improved by deploying sequentially semiseparable matrices to neural networks. This effect is visible for the models with 300 hidden neurons as well as the models with 50 hidden neurons. The optimal state space dimension depends on the dataset at hand. For example, for the MNIST dataset the best performance was achieved with $d = 4$, whereas for the hand postures dataset the optimal state dimension was $d = 1$. The results of our experiments are depicted in Fig. 3. Our observations are in line with previous observations in literature. For other structure classes, it has also been observed that in some cases the prediction accuracy can be increased by using structured weight matrices [9,11,13,19].

The results of our 50 hidden neuron model experiments show that the performance gain of our models is not only due to the standard neural network being overparameterized. We suspect that the latter is the case for the model with 300 neurons trained on the pen-based digits recognition dataset as well as the hand postures dataset. The fact that the small model achieved better test accu-

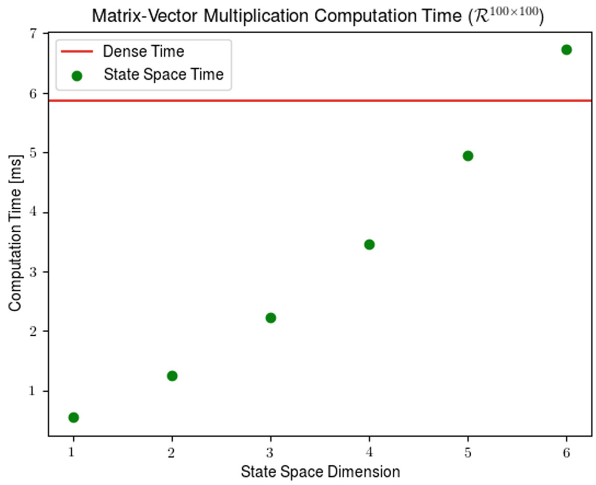

Fig. 4. Comparison of the time needed for computing the matrix-vector product of a 100-dimensional matrix on a STM32F405 microcontroller. When the state dimension is small, the matrix-vector product using a sequentially semiseparable matrix can be performed in a fraction of the time needed for the dense counterpart. However, with increasing state dimension, the time needed for computing the matrix-vector product also increases, finally reaching a break-even point with the dense counterpart.

racies on these datasets suggests that the larger network was overparameterized and thus the training stopped early due to overfitting. In contrast, the models with sequentially semiseparable weight matrix suffered far less from overfitting problems, since these networks inherently comprise fewer parameters. This can, however, not only be attributed to the reduced number of parameters, since the rank 1 weight matrix approach could not consistently outperform the standard neural network.

Interestingly, in our experiments, the inference as well as training times increased for neural networks with sequentially semiseparable weight matrices, even though the number of operations required for inference decreased. This has two reasons. Firstly, we executed our experiments on an AMD Ryzen Threadripper 3990X 64-Core Processor. When computing the matrix-vector product for a standard matrix, the operations could be distributed over 64 available cores. However, computing the matrix-vector product for sequentially semiseparable matrices has fewer parallelization capabilities, since in order to compute the output of timestep $k + 1$, the state of timestep k is required. Therefore, the sequential computation scheme prohibits elaborate parallelization of the operations. The fact that our code is implemented in Python also plays a role here. Our for-loops implemented in Python cannot keep up with the speed of the C-routines used by pytorch. Secondly, computing the computational graph for gradient updates is more complex in our proposed algorithm. As explained in Sect. 4, the training error is propagated through all states. This results in a longer chain of derivatives, similar as obtained in deep neural networks. Constructing and evaluating this computational graph might result in longer training times. Usually, this is not a problem, because for most applications the inference time is the most important factor, and the training time plays a minor role. In summary, we expect longer training times for the proposed method. The reduced number of required operations might not necessarily lead to reduced inference times, since the inference time depends on the targeted hardware and the implementation at hand. However, on hardware with few processing units (like for example microcontrollers), we can expect speed-ups regarding the inference times. We illustrate this by measuring the time needed for computing the matrix-vector multiplication between a matrix $A \in \mathcal{R}^{100\times 100}$ and a random vector on a STM32F405 microcontroller with 168 MHz. Figure 4 shows that the matrix-vector product for the sequentially semiseparable matrix can be computed in a fraction of the time compared to the dense matrix. The computation time increases with increasing state dimension, until the break-even point with the dense computation is reached.

6 Conclusion

We introduced an algorithm for training neural networks with sequentially semiseparable weight matrices. The key idea of the proposed algorithm is to backpropagate the training loss to the matrices generating the structured weight matrix, instead of the entries in the weight matrix directly. By that, it is possible to reduce the training loss while maintaining the structure in the weight matrices throughout training. Moreover, our algorithm can be combined with the standard backpropagation algorithm. This allows for training neural networks comprising structured as well as non-structured weight matrices.

To validate our algorithm, we ran experiments using four real-world datasets. Two key findings resulted from our experiments. First, using structured matrices in neural networks can increase the generalization performance of the network. This finding confirms previous observations from literature, but now applied to sequentially semiseparable weight matrices. Second, computing the product between a sequentially semiseparable matrix and a vector can hardly be parallelized without losing the advantages of reduced required number of operations. Therefore, the structure investigated in this paper is mostly relevant to applications running on non-parallelized hardware such as microcontrollers.

References

1. Alimoglu, F., Alpaydin, E.: Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition. In: Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN 1996). Citeseer (1996)
2. Chandrasekaran, S., Gu, M., Pals, T.: A fast ULV decomposition solver for hierarchically semiseparable representations. SIAM J. Matrix Anal. Appl. **28**(3), 603–622 (2006)
3. Denil, M., Shakibi, B., Dinh, L., De Freitas, N., et al.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems, pp. 2148–2156 (2013)

4. Dewilde, P., Van der Veen, A.J.: Time-Varying Systems and Computations. Springer, New York (1998). https://doi.org/10.1007/978-1-4757-2817-0
5. Fan, Y., Lin, L., Ying, L., Zepeda-Núnez, L.: A multiscale neural network based on hierarchical matrices. Multisc. Model. Simul. **17**(4), 1189–1213 (2019)
6. Gantmakher, F., Krein, M.: Sur les matrices completement non négatives et oscillatoires. Compos. Math. **4**, 445–476 (1937)
7. Gardner, A., Kanno, J., Duncan, C.A., Selmic, R.: Measuring distance between unordered sets of different sizes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 137–143 (2014)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
9. Ioannou, Y., Robertson, D., Shotton, J., Cipolla, R., Criminisi, A.: Training CNNs with low-rank filters for efficient image classification. arXiv preprint arXiv:1511.06744 (2015)
10. Kung, S., Lin, D.: Optimal Hankel-norm model reductions: multivariable systems. IEEE Trans. Autom. Control **26**(4), 832–852 (1981)
11. Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553 (2014)
12. Sindhwani, V., Sainath, T., Kumar, S.: Structured transforms for small-footprint deep learning. In: Advances in Neural Information Processing Systems, pp. 3088–3096 (2015)
13. Tai, C., Xiao, T., Zhang, Y., Wang, X., et al.: Convolutional neural networks with low-rank regularization. arXiv preprint arXiv:1511.06067 (2015)
14. Thomas, A.T., Gu, A., Dao, T., Rudra, A., Ré, C.: Learning compressed transforms with low displacement rank. Adv. Neural. Inf. Process. Syst. **2018**, 9052 (2018)
15. Titti, A., Squartini, S., Piazza, F.: A new time-variant neural based approach for nonstationary and non-linear system identification. In: 2005 IEEE International Symposium on Circuits and Systems, pp. 5134–5137. IEEE (2005)
16. Van Lint, J., Hoogendoorn, S., van Zuylen, H.J.: Accurate freeway travel time prediction with state-space neural networks under missing data. Transp. Rese. Part C: Emerg. Technol. **13**(5–6), 347–369 (2005)
17. Vandebril, R., Van Barel, M., Golub, G., Mastronardi, N.: A bibliography on semiseparable matrices. Calcolo **42**(3), 249–270 (2005)
18. Vandebril, R., Van Barel, M., Mastronardi, N.: Matrix Computations and Semiseparable Matrices: Linear Systems, vol. 1. JHU Press (2007)
19. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems, pp. 2074–2082 (2016)
20. Xie, D., Xiong, J., Pu, S.: All you need is beyond a good init: exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6176–6185 (2017)
21. Zamarreño, J.M., Vega, P.: State space neural network. properties and application. Neural Netw. **11**(6), 1099–1112 (1998)
22. Zhao, L., Liao, S., Wang, Y., Li, Z., Tang, J., Yuan, B.: Theoretical properties for neural networks with weight matrices of low displacement rank. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 4082–4090. JMLR. org (2017)

5.3. Exploiting Structures in Weight Matrices for Efficient Real-Time Drone Control with Neural Networks

Authors: Matthias Kissel, Sven Gronauer, Mathias Korte, Luca Sacchetto, and Klaus Diepold
Journal: Progress in Artificial Intelligence (Proceedings of the 21st EPIA Conference on Artificial Intelligence)
Publisher, publication year: Springer, 2022
Core publication: Yes
Pages: 525–536
Bibliography Entry: [52]
Comment: Reproduced with permission from Springer Nature

Summary

The focus of this paper lies on approximating weight matrices of trained NNs. For that, we consider four structure classes, namely low rank matrices, matrices of low displacement rank, SSS matrices, and products of sparse matrices. For each structure class it is shown how weight matrices can be approximated with structured matrices corresponding to the respective class. We compare the approaches to each other using weight matrices from NNs trained for controlling a quadrotor drone. The experimental setting is the same as in the motivational example introduced in Section 1.2 of this thesis. In our benchmark, we approximate the trained weight matrices using matrices from each structure class and analyze the flying capabilities of the controller based on the approximated NN. The aim of our comparison is to analyze the trade-off between reduction of the number of parameters in the weight matrices and the performance regarding flying robustly in the real world. Our results show that there is structure in the weight matrices, which can be exploited to speed up inference, while still being able to perform the flight maneuvers in the real world.

Own Contributions

- Literature review on related approaches as well as for identification of algorithms that could be used for finding structure in matrices
- Implementation of the algorithms described in the paper
- Execution of experiments in simulation as well as in the real world in collaboration with Sven Gronauer and Mathias Korte
- Analysis, interpretation and discussion of the results

Exploiting Structures in Weight Matrices for Efficient Real-Time Drone Control with Neural Networks

Matthias Kissel( ), Sven Gronauer, Mathias Korte, Luca Sacchetto, and Klaus Diepold

Technical University of Munich, Arcisstr. 21, 80333 Munich, Germany
matthias.kissel@tum.de
https://www.tum.de/en/

Abstract. We consider the task of using a neural network for controlling a quadrotor drone to perform flight maneuvers. For that, the network must be evaluated with high frequency on the microcontroller of the drone. In order to maintain the evaluation frequency for larger networks, we search for structures in the weight matrices of the trained network. By exploiting structures in the weight matrices, the propagation of information through the network can be made more efficient. In this paper, we focus on four structure classes, namely low rank matrices, matrices of low displacement rank, sequentially semiseparable matrices and products of sparse matrices. We approximate the trained weight matrices with matrices from each structure class and analyze the flying capabilities of the approximated neural network controller. Our results show that there is structure in the weight matrices, which can be exploited to speed up the inference, while still being able to perform the flight maneuvers in the real world. The best results were obtained with products of sparse matrices, which could even outperform non-approximated networks with the same number of parameters in some cases.

Keywords: Neural control · Structured matrices · Fast inference

1 Introduction

Neural networks are universal function approximators [3]. Therefore, they are used in an increasing number of areas. One such area is neural drone control, where a neural network is used to control an autonomously flying drone. In our case, we focus on performing flight maneuvers with a Crazyfly 2.1 quadrotor drone[1]. For that, we train a neural network in simulation using reinforcement learning, and then deploy the network to the real-world (sim-to-real). This training pipeline is explained in detail in [9].

In this paper, we focus on implementing the flying policy in form of the neural network efficiently on the drone. For flying robustly, the neural network

[1] https://www.bitcraze.io/products/crazyflie-2-1/.

G. Marreiros et al. (Eds.): EPIA 2022, LNAI 13566, pp. 525–536, 2022.
https://doi.org/10.1007/978-3-031-16474-3_43

must be evaluated on the drone with a high frequency. This is challenging, since we require that all calculations are performed on board of the drone, i.e. on a STM32F405 microcontroller with 168 MHz.

Since we use densely connected layers in our policies, the dominant cost for propagating information through the network arises in form of matrix-vector multiplications. For example, to compute the matrix-vector product of a matrix $\mathcal{R}^{n \times n}$ requires $\mathcal{O}(n^2)$ operations. In this paper, we aim to reduce this order of required operations by exploiting structures in the weight matrices of the network. If the weight matrices possess certain structures, the order of required operations needed shrinks to the subquadratic domain. For example, if the weight matrix has many zero entries, the propagation can be made more efficient by exploiting the sparsity in the matrix. For sparse matrices, this results in $\mathcal{O}(k)$ operations for matrices with k nonzero entries. In Sect. 3, we present other matrix structures which can be used for making the inference more efficient.

Our contribution is two fold. First, we introduce several methods which can be used for approximating given weight matrices with structured matrices. Second, we perform extensive experiments for finding structures in the trained weight matrices of a neural network used for neural drone control. This leads to findings regarding the best approximation methods and approximation norms.

The rest of this paper is organized as follows. First, we review approaches in literature that have been used to make neural network evaluation faster. Here, we mainly focus on exploiting structures in the weight matrices of neural networks. In the subsequent section, we introduce the methods we used for approximating weight matrices in neural networks. In Sect. 5, we show the results of our experiments in a simulation environment as well as on our drone flying in the real-world. Finally, we summarize our findings and give a conclusion.

2 Literature Review

There are several approaches in literature targeting to reduce the time required for neural network inference [18]. These include for example optimizing the dataflow [18], quantization techniques [15], using specialized hardware [8], as well as knowledge distillation [10]. For example, many approaches in neural network training or post-processing aim at producing sparse weight matrices [1,14]. Using these approaches, the number of operations needed for computing the matrix-vector multiplication decreases, as explained in the introduction.

In this paper, we focus on structured matrices. Structured matrices can be described with less than $\mathcal{O}(n^2)$ parameters. In contrast to sparse matrices, structured matrices must not contain zeros. Instead, their entries have a relationship to each other, which we denote as *structure*. The research field of structured matrices is fragmented, which means that there are many different special cases of matrix structures. In this paper, we are interested in structures for which the multiplication of the structured matrix with an arbitrary vector requires subquadratic order of operations. Therefore, we focus on four structure classes, namely low rank matrices, matrices of low displacement rank [16], sequentially semiseparable matrices [7] and products of sparse matrices [4,5,13] (note that the product of sparse matrices is not sparse in general). An introduction to each of these classes is given in Sect. 3.

There are several approaches in literature, where structured matrices have been applied to neural networks. For example, weight matrices in neural networks have been replaced by structured matrices to be trained using the backpropagation algorithm afterwards [4,5,17,19,20]. To the best of our knowledge, there is no extensive comparison of approximating a given weight matrix in a neural net with different kind of structures yet.

3 Methodology

Our aim is to reduce the number of operations required for inference using a trained neural network. For that, we search for structures in the *trained* weight matrices in order to replace the original weight matrices with structured counterparts. Since the application of this paper is neural drone control, our trained network is able to control a quadrotor drone to fly a specific maneuver (in this paper, we investigate the task of flying in a circle). We start from a given neural network J, which is a composition of layer mappings

$$J(u) = (\mathcal{L}_r \circ \cdots \circ \mathcal{L}_1)(u), \tag{1}$$

where the neural network consists of r layers and $\mathcal{L}_i$ is the mapping of the i^{th} layer. We focus on densely connected feed-forward neural networks, i.e. the layer mappings are of the form

$$\mathcal{L}_i(u) = \sigma(W_i u + \theta_i), \tag{2}$$

where W_i is a weight matrix, θ_i are the biases of the respective layer, and σ the activation function of the layer, which is applied element-wise to its inputs.

Given the trained neural network J, we search for structure in its weight matrices W_i. We say that W_i approximately possesses a certain structure, if there is a matrix $\hat{W}_i$ which has the desired structure and

$$||W_i - \hat{W}_i||_N < \epsilon. \tag{3}$$

$||\cdot||_N$ is a matrix norm (for example the Frobenius norm) and ϵ is the maximum error which we tolerate. If the tolerance is chosen too big, the approximated network does not fly in the real world. On the opposite side, if ϵ is chosen too small, we might not find a structured matrix $\hat{W}$ which fulfills the requirements. In practice, we usually do not know the right tolerance beforehand. Therefore, our approach is to find the best approximation $\hat{W}_i$ for a given weight matrix W_i with respect to different structure classes. Afterwards, the approximated weight matrix W_i is evaluated in terms of number of parameters and flying capabilities of the overall network.

We investigate four matrix structure classes in order to find approximations for our given weight matrices W_i. These four structure classes are introduced in the following. The code used for runnning our experiments can be found online[2].

3.1 Low Rank Matrices

The most straightforward matrix structure we investigate are low rank matrices. If our weight matrix $W_i \in \mathcal{R}^{m \times n}$ has rank $r < \min(m, n)$, it can be expressed as

$$W_i = GH, \tag{4}$$

with $G \in \mathcal{R}^{m \times r}$ and $H \in \mathcal{R}^{r \times n}$. Since W_i most likely does not possess a low rank, we are interested in finding a low rank approximation for W_i.

We follow two independent approaches for finding low rank approximations. The first approach uses the singular value decomposition (SVD) of W_i as

$$W_i = USV^T. \tag{5}$$

Using the SVD, we can find the best 2-norm as well as Frobenius norm rank r approximation for W_i by setting all singular values σ_j with $j > r$ to zero

$$\hat{S}_{j,j} = \begin{cases} \sigma_j & \text{if} j \leq r \\ 0 & \text{else} \end{cases}. \tag{6}$$

This results in $G = U\sqrt{\hat{S}}$ and $H = \sqrt{\hat{S}}V^T$. We are also interested in the approximation result if we target other norms than the 2-norm or the Frobenius norm. Therefore, we have a second approach, which consists of glorot-uniform randomly initializing the matrices G and H. Both matrices are then optimized using gradient descent to minimize

$$\min ||W_i - GH||_N, \tag{7}$$

where N is the norm we aim to minimize (-1, 1, -2, inf, $-$inf or the nuclear norm). We use Adam [11] with the pytorch[3] standard hyperparameters and initial learning rate of 0.1 as step-size optimizer. Each optimization is repeated 5 times taking the best approximation result in order to account for the random initialization of the initial G and H. We train until the loss reduction between two optimization steps is smaller than $1e-4$.

3.2 Matrices of Low Displacement Rank

Matrices of low displacement rank [16] build on the notion that a matrix might possess low rank after displacing its entries. The displacement rank can for example be measured using the Sylvester type operators

$$L(W_i) = \nabla_{A,B}(W_i) = AW_i - W_iB = GH. \tag{8}$$

[2] https://github.com/MatthiasKi/drone_structures.

[3] https://pytorch.org/.

Here, A and B are fixed operator matrices. $W_i \in \mathcal{R}^{m \times n}$ is said to have a low displacement rank, if $L(W_i)$ has low rank, i.e. $G \in \mathcal{R}^{m \times r}$ and $H \in \mathcal{R}^{r \times n}$ with $r < \min(m, n)$.

In this paper, we follow the approach from Thomas et al. [19] and learn the operator matrices A and B jointly with the matrices G and H. For that, we parameterize the operator matrices as tridiagonal plus corner matrices, which includes many well-known standard operators. Therefore, our parameterization inter alia contains toeplitz-like, hankel-like, vandermonde-like and cauchy-like matrices [19].

We formulate the problem of finding suitable A, B, G and H matrices as optimization problem

$$\min_{A,B,G,H} ||W_i - decompress(A, B, G, H)||_N, \tag{9}$$

where N is the norm we want to minimize (chosen from -1, 1, -2, 2, inf, $-$inf, nuclear or Frobenius norm). The $decompress()$ method recovers the matrix $\hat{W}_i$ from the determined displacement operators

$$\begin{aligned} \hat{W}_i &= decompress(A, B, G, H) \\ &= \Sigma_{i=1}^{r} \mathcal{K}(A, G_i)\mathcal{K}(B^T, H_i^T)^T, \end{aligned} \tag{10}$$

which has a displacement rank at most $2r$ [19]. Here $\mathcal{K}(A, v)$ denotes the $n \times n$ Krylov matrix where the i^{th} column is determined as $A^i v$. We denote the i^{th} column of G with G_i (H_i^T respectively). Note that this approach can only be used for approximating square matrices W_i.

We determine suitable A, B, G and H matrices using stochastic gradient descent. Each approximation run consists of three subsequent optimizations with different learning rates (1, 0.1 and 0.01), whereas each optimization run continues until the minimization loss can not be improved more than $1e - 5$. Each approximation run is repeated 5 times (taking the best approximation result) in order to account for the random initialization of the A, B, G and H matrices.

3.3 Sequentially Semiseparable Matrices

Sequentially Semiseparable Matrices originate from Time Varying Systems theory [7]. This structure naturally arises when describing the input-output behavior of a time varying system, and is defined as

$$\begin{aligned} W_i = D &+ C(I - ZA)^{-1}ZB \\ &+ G(I - Z^T E)^{-1} Z^T F. \end{aligned} \tag{11}$$

Here, I is the identity matrix and Z is a down-shift matrix

$$Z = \begin{pmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & 0 \\ 0 & & 1 & 0 \end{pmatrix}. \tag{12}$$

A, B, C, D, E, F and G are block-diagonal matrices, each comprising of n matrices

$$A = diag([A_1, \ldots, A_n]) \tag{13}$$

(B, C, D, E, F and G matrices respectively). In the context of time-varying system theory, the matrices $A, \ldots, G$ define the behavior of a time-varying system. For example, A maps the previous state of the system to the next state and B maps the previous state to the current output.

In order to find an approximation of W_i which possesses the sequentially semiseparable structure, we use the time-varying system realization theory in combination with balanced model reduction [7,12]. We describe our approach for square matrices $W_i \in \mathcal{R}^{n \times n}$ for better illustration. This approach can be straightforwardly extended to non-square cases (for more details we refer to our code). We set the input and output dimensions to 1, which results in $D_j \in \mathcal{R}^{1 \times 1}$. Then, we determine the biggest possible state dimension d which still has less than the allowed number of parameters. This results in realization matrix shapes $A_k \in \mathcal{R}^{d \times d}$, $B_k \in \mathcal{R}^{d \times 1}$, $C_k \in \mathcal{R}^{1 \times d}$, $D_k \in \mathcal{R}^{1 \times 1}$, $E_k \in \mathcal{R}^{d \times d}$, $F_k \in \mathcal{R}^{d \times 1}$ and $G_k \in \mathcal{R}^{1 \times d}$ for $k = 1, \ldots, n$. These realization matrices are obtained by the standard realization approach [7], but cutting out all singular values σ_l for $l > d$ from the Hankel matrices obtained during realization.

3.4 Products of Sparse Matrices

As shown in Sect. 2, there exist many approaches for promoting sparsity in the weight matrices of neural networks. We build on this theory, but in contrast to most existing literature, we investigate *products of sparse matrices*. Recently, interesting results about weight matrices represented as product of sparse matrices have been reported [4–6]. In general, the product of sparse matrices is not sparse. Moreover, the notion of sparsity and structure in linear maps seems to be fundamentally linked [4,6], which leads to the conclusion that all efficient matrix-vector multiplication algorithms can be factorized into products of sparse matrices [5].

In order to approximate a given weight matrix $W_i \in \mathcal{R}^{m \times n}$ with a product of sparse matrices F_j, we minimize

$$||W_i - \prod_{j=1}^{k} F_j||_F, \tag{14}$$

with

$$\sum_{j=0}^{k} nnz(F_j) \leq \psi, \tag{15}$$

where $nnz()$ denotes the number of nonzero elements in a matrix and ψ is the maximum number of nonzero elements in the product. We treat the number of sparse matrices k as a hyper parameter and fix the shapes of F_j

$$F_j \begin{cases} \in \mathcal{R}^{m \times max(m,n)} & \text{if } j = 1 \\ \in \mathcal{R}^{max(m,n) \times n} & \text{if } j = k \\ \in \mathcal{R}^{max(m,n) \times max(m,n)} & else \end{cases} . \tag{16}$$

We do not need to optimize the shapes of the factors if they are chosen large enough, because smaller shapes are contained as submatrices of larger shapes.

We use the algorithm proposed by Magoarou and Gribonval [13] in order to find F_j. They proposed to use the Proximal Alternating Linearized Minimization (PALM) [2] algorithm for iterative factorization of a given matrix into sparse factors. The PALM algorithm updates the factors of the sparse product using projected gradient descent steps (in our case we use a projection onto matrices with prescribed sparsity). Based on this, Magoarou and Gribonval proposed to follow a hierarchical approach, where they subsequently add sparse factors to the product in order to approximate a given matrix.

In our experiments, we used this hierarchical approximation algorithm based on PALM for approximating the W_i matrices. We repeated the approximation for different hyperparameters in order to find the best combination for a given weight matrix. For that, we tried different numbers of factors in the product ($k = 1, \ldots, 9$) and different distributions of the number of nonzero elements across the factors in the product. The different distributions were generated by fixing the number of nonzero elements in the last factor $nnz(F_k)$ and determining the number of nonzero elements following a linear

$$nnz(F_j) = floor(\alpha + mj), \tag{17}$$

or exponential distribution

$$nnz(F_j) = floor(\alpha e^{mj}). \tag{18}$$

The parameters m and α can be determined using $nnz(F_k)$ and the constraint given in Eq. 15. In each experiment, we tried different values for $nnz(F_k)$, equally distributed in the range $[0.1\psi, 0.9\psi]$. If $k = 1$, the product over sparse matrices reduces to a single sparse matrix. In this case, we skip the hierarchical optimization scheme and simply use the ψ elements with highest absolute value in the resulting sparse matrix.

4 Experimental Setup

In our experiments, we approximate the weight matrices of two 2-hidden-layered neural networks. We chose to investigate models with 6 and 30 hidden neurons respectively, in order to compare the approximation results for different model sizes. Both models are able to fly in the real world (whereas the bigger model

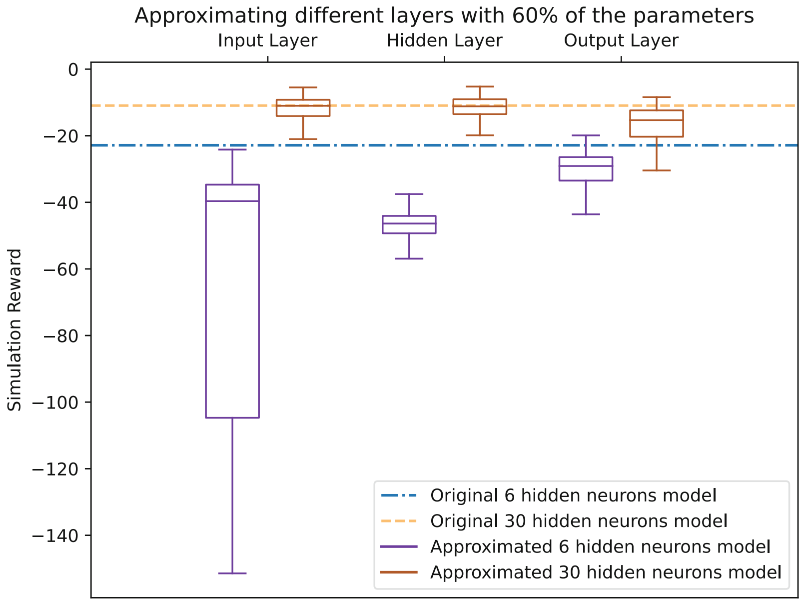

Fig. 1. Comparison of the rewards obtained after approximating certain layers of a neural network for different hidden layer sizes. The approximation tends to be better for networks containing larger weight matrices. For such networks, approximating the input and/or hidden layer weight matrix yields best performance. However, for smaller networks, the effect that approximation errors in early layers of the network might get amplified in later layers becomes apparent. For these networks, it might be better to remove parameters in deeper layers (for example the output layer).

is more robust). For evaluating the models, the original weight matrices are substituted by approximated counterparts in the neural network. The resulting network is evaluated in terms of its flying capabilities in the simulation. For that, we measure the cumulated reward obtained during flying, which was used to train the original model. Hence, the evaluation metric is independent of the matrix norm used during approximation.

Each model is evaluated in terms of its cumulated reward over 500 test trajectories in simulation. The reward takes into account the deviation of the drone position to the ideal position for flying the maneuver, plus terms penalizing undesired flying behaviors such as shaking or drastically changing the motor outputs frequently. We report the mean and the standard deviation of the obtained reward. A reward higher than −25 usually means that the drone is able to fly in the real world (the reward is optimally around −10). If the reward is in the range [−80, −25], this means that the neural network is occasionally able to perform the flight maneuver, but also crashes sometimes. Rewards lower than −80 lead to crashes of the drone in the real world (as well as in simulation) for most times.

5 Results

The approximation results tend to be better for bigger weight matrices than for smaller ones (as shown in Fig. 1). This particularly affects approximating

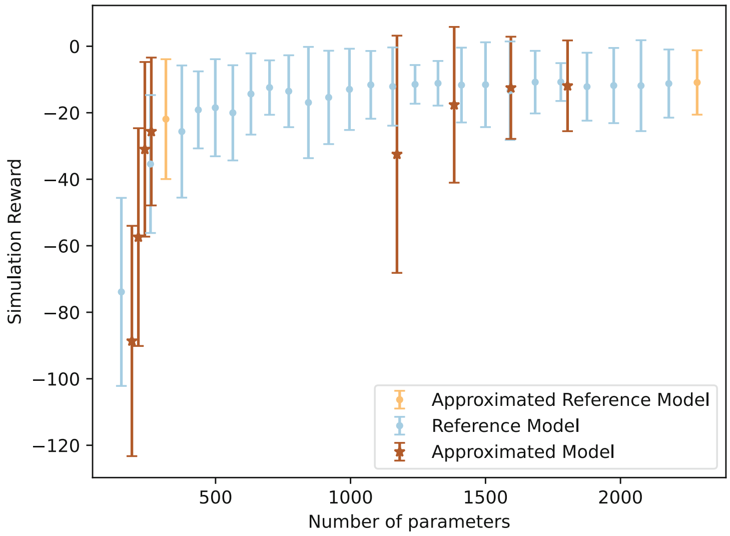

Fig. 2. Rewards achieved by our approximated models compared to models with different number of hidden neurons trained from scratch. Models where we removed a small number of parameters achieved similar rewards like the non-approximated counterparts. If too many, parameters were removed, the models trained from scratch usually outperformed the approximated models.

the output matrix of the network, which leads to worse results compared to the other matrices (depending on the number of hidden neurons). Since our network produces 4 outputs, the size of the output matrix has usually an order of magnitude less parameters. This results in $4z$ parameters (with z being the hidden layer size), compared to z^2 parameters in the hidden layer matrix and $40z$ parameters in the input layer matrix. The network with 6 hidden neurons is less affected by this, because here the hidden layer does not have significantly more parameters than the output layer. Instead, another effect plays a major role here: Errors introduced by approximating the input layer can be amplified while being propagated through the other layers. Hence, in the case of the small network, removing parameters from deeper layers results in better performance.

Therefore, we suspect that more structures are present in the bigger weight matrices. This might be due to overparameterization of the bigger network. Figure 2 shows that a hidden layer size with more than 12 neurons does not result in significant improvements of the obtained cumulated reward. Therefore, we suspect that the model with 30 hidden neurons is overparameterized, which might contribute to the good approximation capabilities.

In order to investigate the tradeoff between parameter reduction and flight capability, we approximated both models with different number of parameters. In the 30 hidden neuron model both, the input layer as well as the hidden layer weight matrix, are approximated with products of sparse matrices. In the 6 hidden neuron model, only the input layer is approximated. The weight matrices are approximated using products of sparse matrices, since this resulted in the best performance. Figure 2 shows the results for the approximated models. Surprisingly, there were even approximated models which performed better than

Table 1. Time needed for inference on the drone microcontroller. The models with 100% parameter share refer to the original models. In the other models, weight matrices have been approximated with products of sparse matrices.

# Neurons	6	6	6	6	30	30	30	30	30
Parameter share [%]	100	80	70	60	100	80	70	60	50
Mean inference time [ms]	0.058	0.068	0.054	0.05	0.4	0.44	0.38	0.34	0.3

their non-approximated counter parts with the same number of parameters. For the model with 30 neurons in the hidden layer, approximation of the weight matrices led to similar performance like training a model with the same number of parameters from scratch, if not too many parameters were removed.

Our approximated models were able to control the drone to fly the circle maneuver even in the real-world. We recorded videos to show the flying performance of the different models compared to the original models[4]. Moreover, we measured the time needed for inference on the drone microcontroller. The results are shown in Table 1. It can be seen, that the time required for inference decreases if there are fewer parameters in the network. However, the computations with sparse matrices also produce an overhead compared to the standard matrix-vector multiplication. Thus, the approximation is only worthwhile in terms of inference time reduction if a certain reduction of the parameters is reached.

A comparison of the performance of the different approximation methods is shown in Fig. 3. Here, we compare the cumulated reward obtained by a model with 30 hidden neurons, for which we approximated the hidden layer weight matrix with 80% of the original number of parameters. It can be seen that the Frobenius and 2-Norm approximation led to the best results for the low rank approximation as well as the low displacement rank approximation methods. Moreover, approximations based on the nuclear norm led to acceptable approximations as well as the 1-norm approximation used in the low rank approach. Using the -1, -2, inf or $-\inf$ norms led to consistently bad results.

We would like to point out that for smaller matrices or fewer number of parameters the low displacement rank approach as well as the sequentially semiseparable matrices approach tended to produce poor results. We suspect that this is due to the fact that these matrix structures usually only have advantages for large matrices. Therefore, results might be different for larger neural networks than the ones used in our experiments. However, in some rare cases, the sequentially semiseparable approximation performed very good compared to the other methods.

[4] https://youtu.be/PVaTnagaUzs.

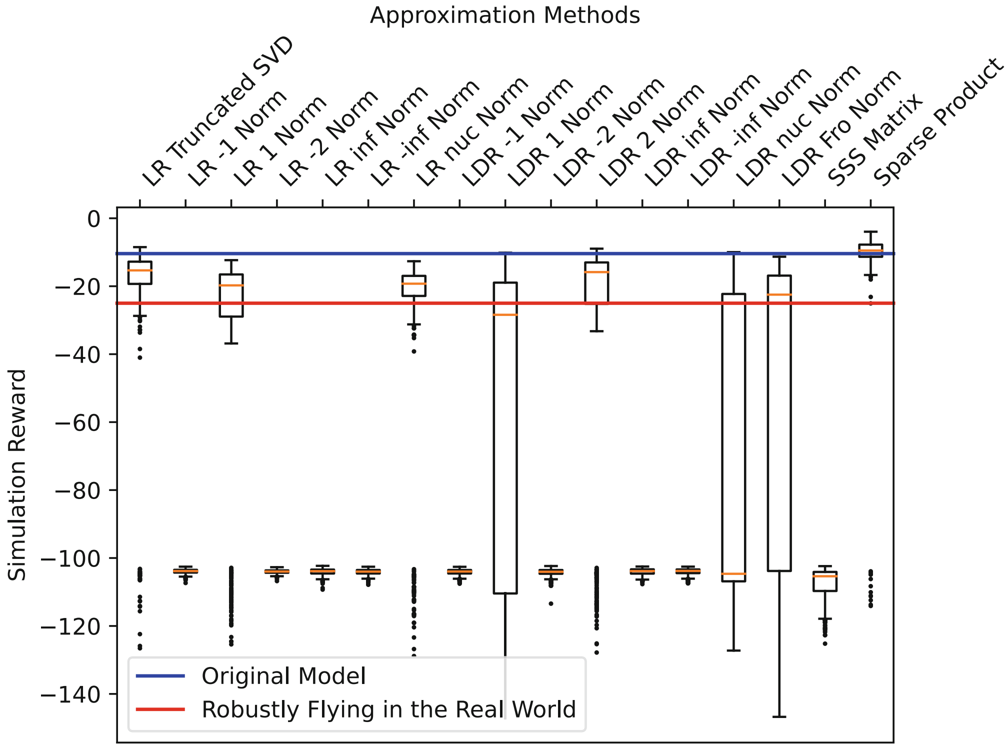

Fig. 3. Comparison of the achieved reward by approximating the hidden layer weight matrix of a model with 30 hidden neurons with 80% of it's parameters. The product of sparse matrices yields the best results. Regarding the low rank and low displacement rank approaches, using the 2-Norm or the Frobenius Norm as optimization objective led to the best results. The nuclear and the 1-Norm also achieved acceptable approximation results, whereas approximating the weight matrix targeting other norms led to bad simulation rewards.

6 Conclusion

We analyzed the weight matrices of a trained neural network used for neural drone control. For that, we approximated the trained weight matrices of the network with structured matrices using four different approaches. Our results showed that the weight matrices possess structure, which can be exploited to speed up the inference. Approximating the weight matrices with products of sparse matrices showed to be the most promising approach in our experiments. With this approach, we could achieve approximations with fewer parameters, which almost had the same flying capabilities as the original model. In the case of very small networks, the approximation could even outperform neural networks with the same number of parameters trained from scratch.

References

1. Blalock, D., Ortiz, J.J.G., Frankle, J., Guttag, J.: What is the state of neural network pruning? arXiv preprint arXiv:2003.03033 (2020)
2. Bolte, J., Sabach, S., Teboulle, M.: Proximal alternating linearized minimization for nonconvex and nonsmooth problems. Math. Program. **146**, 459–494 (2013). https://doi.org/10.1007/s10107-013-0701-9

3. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989)
4. Dao, T., Gu, A., Eichhorn, M., Rudra, A., Ré, C.: Learning fast algorithms for linear transforms using butterfly factorizations. In: International Conference on Machine Learning, pp. 1517–1527. PMLR (2019)
5. Dao, T., et al.: Kaleidoscope: an efficient, learnable representation for all structured linear maps. arXiv preprint arXiv:2012.14966 (2020)
6. De Sa, C., Cu, A., Puttagunta, R., Ré, C., Rudra, A.: A two-pronged progress in structured dense matrix vector multiplication. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1060–1079. SIAM (2018)
7. Dewilde, P., Van der Veen, A.J.: Time-Varying Systems and Computations. Springer, New York (1998). https://doi.org/10.1007/978-1-4757-2817-0
8. Furber, S.B., Galluppi, F., Temple, S., Plana, L.A.: The spinnaker project. Proc. IEEE **102**(5), 652–665 (2014)
9. Gronauer, S., Kissel, M., Sacchetto, L., Korte, M., Diepold, K.: Using simulation optimization to improve zero-shot policy transfer of quadrotors. arXiv preprint arXiv:2201.01369 (2022)
10. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
11. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
12. Kung, S., Lin, D.: Optimal Hankel-norm model reductions: multivariable systems. IEEE Trans. Autom. Control **26**(4), 832–852 (1981)
13. Le Magoarou, L., Gribonval, R.: Flexible multilayer sparse approximations of matrices and applications. IEEE J. Sel. Top. Signal Process. **10**(4), 688–700 (2016)
14. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. Adv. Neural Inf. Processing Syst. **2** (1989)
15. Lee, E.H., Miyashita, D., Chai, E., Murmann, B., Wong, S.S.: LogNet: energy-efficient neural networks using logarithmic computation. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5900–5904. IEEE (2017)
16. Pan, V.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Springer, Boston (2001). https://doi.org/10.1007/978-1-4612-0129-8
17. Sindhwani, V., Sainath, T.N., Kumar, S.: Structured transforms for small-footprint deep learning. arXiv preprint arXiv:1510.01722 (2015)
18. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: a tutorial and survey. Proc. IEEE **105**(12), 2295–2329 (2017)
19. Thomas, A.T., Gu, A., Dao, T., Rudra, A., Ré, C.: Learning compressed transforms with low displacement rank. Adv. Neural. Inf. Process. Syst. **2018**, 9052 (2018)
20. Zhao, L., Liao, S., Wang, Y., Li, Z., Tang, J., Yuan, B.: Theoretical properties for neural networks with weight matrices of low displacement rank. In: International Conference on Machine Learning, pp. 4082–4090. PMLR (2017)

5.4. Deep Convolutional Neural Networks with Sequentially Semiseparable Weight Matrices

Authors: Matthias Kissel and Klaus Diepold
Journal: ESANN 2022 Proceedings (30th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning)
Publisher, publication year: i6doc, 2022
Core publication: Yes
Bibliography Entry: [48]

Summary

Modern CNNs comprise millions of parameters. Therefore, the use of these networks requires high computing and memory resources. In this paper, we propose to reduce these resource requirements by using structured matrices. For that, we replace weight matrices of the fully connected classifier part of several pre-trained CNNs by SSS Matrices. By that, fewer parameters are required to define the weight matrices of these layers, and the time required for computing the product between the weight matrix and an input vector can be reduced. In our experiments, we compare the prediction performance of NNs with standard weight matrices and rank one weight matrices to NNs with SSS weight matrices. I use the results of these experiments in this thesis to show that the choice of the structure class used in NNs does have an impact on the achieved test accuracy. Furthermore, we show in the paper that the combination of approximating pretrained weight matrices with SSS matrices followed by gradient-descent based training leads to the best prediction results (compared to just approximating or training from scratch).

Own Contributions

- Development and formulation of the algorithms in collaboration with Klaus Diepold
- Design, implementation and execution of the experiments mentioned in the paper
- Analysis and discussion of the results

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

Deep Convolutional Neural Networks with Sequentially Semiseparable Weight Matrices

Matthias Kissel and Klaus Diepold

Technical University of Munich - Chair of Data Processing
Arcisstr. 21 - 80333 Munich - Germany

Abstract. Modern Convolutional Neural Networks (CNNs) comprise millions of parameters. Therefore, the use of these networks requires high computing and memory resources. We propose to reduce these resource requirements by using structured matrices. For that, we replace weight matrices of the fully connected classifier part of several pre-trained CNNs by Sequentially Semiseparable (SSS) Matrices. By that, the number of parameters in these layers can be reduced drastically, as well as the number of operations required for evaluating the layer. We show that the combination of approximating the original weight matrices with SSS matrices followed by gradient-descent based training leads to the best prediction results (compared to just approximating or training from scratch).

1 Introduction

Modern Convolutional Neural Networks (CNNs) consist of many layers and millions of parameters. By that, they are able to achieve remarkable results in image-based problem domains, like image recognition [1, 2]. However, as the number of parameters increases, so does the computational effort required for deploying the network. This is especially a problem for applications targeting mobile devices or embedded hardware like microcontrollers. For these applications, the use of modern CNN architectures is often not possible due to insufficient computing and / or memory resources.

Deep CNNs are often designed similarly. First, there is a feature extractor part, which consists of convolutional and pooling layers. After the feature extractor part, the activations are flattened and put into the classifier part. The classifier usually consists of fully-connected feed-forward layers. A large part of the parameters of the network is typically located in the classifier part, since the parameters in the convolutional filters are shared. As a result, evaluating the classifier part involves performing large matrix-vector multiplications.

Our goal is to reduce the resources needed in the classifier part of deep CNNs. For that, we focus on the last (i.e. fully-connected) layer of the network. Propagating information through this layer requires $\mathcal{O}(nm)$ operations for a weight matrix $W \in \mathcal{R}^{n \times m}$. This order of magnitude can be reduced to the subquadratic domain if the weight matrix of the layer has a specific structure. Particularly, we are interested in using Sequentially Semiseparable (SSS) matrices as weight matrices in neural networks. This matrix structure typically arises when describing linear time-varying systems [3].

Our contribution is two-fold. First, we investigate the effect of replacing the last weight matrix of several state-of-the art CNN models with a SSS matrix. By

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

that, we can analyze the trade-off between number of parameters and prediction accuracy of the overall recognition model. Second, we study the influence of the method used to bring the structure into the neural network. Here, we compare the achieved prediction performance of different approaches like approximating the weight matrix, training an SSS weight matrix from scratch, or the combined approach of approximation and training.

The rest of this paper is organized as follows. We first give an overview over approaches in literature, which use structured matrices in the context of neural networks. Subsequently, we introduce the methods we use to bring the SSS structure into the neural network. In Section 4, we present and discuss our experimental results. Finally, we draw a conclusion.

2 Literature Review

Several approaches in literature proposed the use of structured matrices in neural networks. In this context, matrices of low displacement rank are often used [4]. Prominent representatives of this structure class are Toeplitz matrices, which are connected to CNNs. Other approaches focused on Toeplitz-like weight matrices [5], or trained the operator matrices together with the low-rank components end-to-end in a neural network [6]. Moreover, it has been shown that the universal approximation theorem holds for neural networks with weight matrices of low displacement rank [7].

Other matrix structures used in neural networks are hierarchical matrices [8] and products of sparse matrices [9]. For example, Fan et al. [10] proposed to use hierarchical matrices in neural networks. Later, they extended their approach to $\mathcal{H}^2$ matrices [11]. Several authors proposed to use products of sparse matrices (particularly butterfly matrices) in neural networks [9, 12, 13, 14]. The idea here is that the product of sparse matrices is not sparse in general. Therefore, many dense matrices can be represented as a product of sparse matrices.

3 Methods

Our goal is to replace a weight matrix $W \in \mathcal{R}^{n \times m}$ from a trained neural network with an SSS matrix $\hat{W}$. The SSS matrix has the form

$$\hat{W} = D + C(I - ZA)^{-1}ZB + G(I - Z^T E)^{-1}Z^T F. \quad (1)$$

Here, I is the identity matrix and Z is a down-shift matrix containing ones on the subdiagonal and zeros everywhere else. A, B, C, D, E, F and G are block-diagonal matrices, where each matrix contains p sub-matrices

$$A = diag([A_1, \dots, A_p]) \quad (2)$$

(B, C, D, E, F and G matrices respectively). This structure naturally arises when describing time-varying systems [3], where matrices $A, \dots, G$ explain the behavior of a system. For example, C maps the state of the system to the

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

output, and D maps the inputs of the system to the outputs. Note that the dimensions of the A_k, B_k, C_k, D_k, E_k, F_k and G_k matrices can change for different $k = 1, \ldots, p$. This is due to the fact that input, state and output dimensions might change over time.

We explore two approaches for replacing a given weight matrix with an SSS matrix. The first approach starts from a randomly initialized SSS matrix, which is trained using *Backpropagation through States* [15] (this is a data-driven approach). In contrast, no training data is required for the second approach. Instead, the original weight matrix is approximated using a model order reduction method. For both approaches, suitable dimensions for $A_k, \ldots, G_k$ need to be found. We set these dimensions with the aim to achieve a uniform distribution of the input and output dimensions in the SSS representation. This means, that for a given weight matrix $W \in \mathcal{R}^{n\times m}$, which is to be approximated with an SSS matrix with p stages, the resulting input dimensions $dim(u_k)$ are

$$dim(u_k) = \begin{cases} floor(\frac{m}{p}) + 1 & for \quad k \leq m - floor(\frac{m}{p})p, \\ floor(\frac{m}{p}) & else \end{cases} \tag{3}$$

(output dimensions analogously). The dimension of the states d_k is fixed to be constant for all k ($d_k = d$ for all k). We treat d as a hyper parameter to control the number of parameters available in the SSS matrix.

We use *Backpropagation through States* in order to train SSS weight matrices. The key idea of the algorithm is to derive the training loss $\mathcal{L}$ with respect to the parameters of the structure, *not* with respect to the entries of the resulting weight matrix $\hat{W}_{i,j}$. This results in gradients of the form

$$\frac{\delta\mathcal{L}}{\delta A_k}, \frac{\delta\mathcal{L}}{\delta B_k}, \frac{\delta\mathcal{L}}{\delta C_k}, \frac{\delta\mathcal{L}}{\delta D_k}, \frac{\delta\mathcal{L}}{\delta E_k}, \frac{\delta\mathcal{L}}{\delta F_k}, \frac{\delta\mathcal{L}}{\delta G_k}. \tag{4}$$

We compute these gradients using automatic differentiation as provided by the pytorch machine learning framework[1]. The other steps of the training procedure remain the same as in standard neural network training.

In order to approximate a given weight matrix with an SSS matrix, we use a model order reduction method [3, 16]. This is based on the standard approach for finding a balanced state-space realization for a given transfer operator (which is the original weight matrix in our case). As part of the realization algorithm, the Hankel matrices $\mathcal{H}_i$ of the operator are decomposed into observability and controllability matrices ($\mathcal{O}_i$ and $\mathcal{C}_i$ respectively) using the Singular Value Decomposition

$$\mathcal{H}_i = U_i S_i V_i^T, \quad \mathcal{O}_i = U_i\sqrt{S_i}, \quad \mathcal{C}_i = \sqrt{S_i}V_i^T. \tag{5}$$

At this step, we cut out the smallest singular values until the realization has the desired state dimension (d in our case). By that, we obtain a realization $\hat{W}$, which performs similar to the original weight matrix W, but with a reduced amount of parameters. This procedure is called balanced model reduction for time-invariant systems [3].

[1] https://pytorch.org/

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1.
Available from http://www.i6doc.com/en/.

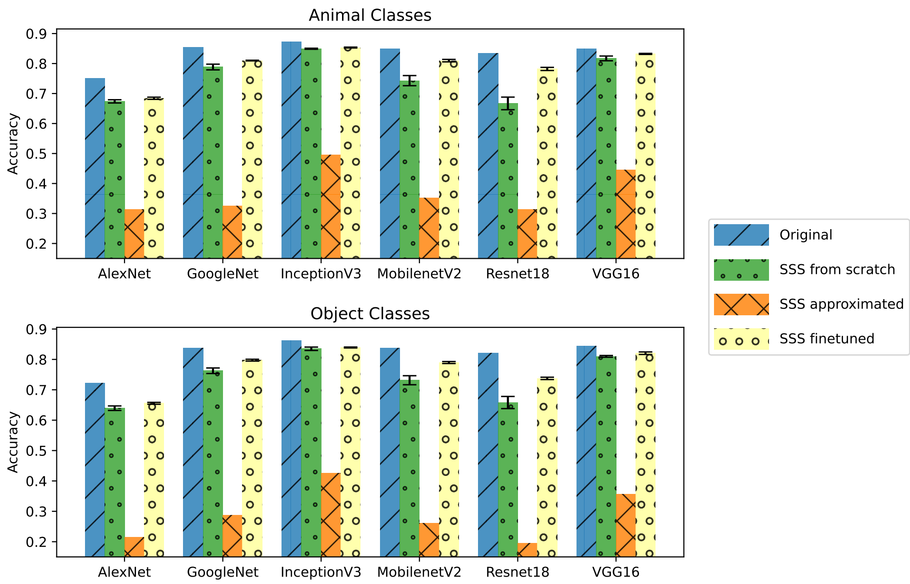

Fig. 1: Approximation with subsequent training led to the best results for all models (the resulting SSS matrix comprises 20% of the parameters).

4 Results

We conduct experiments with pretrained deep convolutional neural networks obtained from pytorch, namely *AlexNet*, *VGG16*, *ResNet18*, *InceptionV3*, *MobilenetV2*, and *GoogleNet*. The models are pretrained on the ImageNet 2012 dataset for image recognition [17]. For our experiments, we selected two subsets of images from the overall Imagenet dataset. Each subset comprises 100 classes from the original dataset (animals and objects), whereas each class comes with approximately 1000 training images and 50 validation images. By that, we can compare the effects on two distinct datasets for several models. We report the mean and standard deviation of the accuracy on the ImageNet validation set. This set has not been used in our training procedure (but it might have been used for pretraining the models).

We replace the weight matrix of the last, fully-connected layer with an SSS matrix. For this, we compare three approaches: Approximating the weight matrix, approximation followed by training, and training the sequentially semiseparable matrix from scratch (after random initialization). Our code used for conducting the experiments can be found online[2].

Replacing the original weight matrix with an SSS matrix obtained from balanced model reduction consistently led to bad prediction accuracy for the resulting model in all experiments. However, the combined approach of approximation

[2]https://github.com/MatthiasKi/structurednets

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

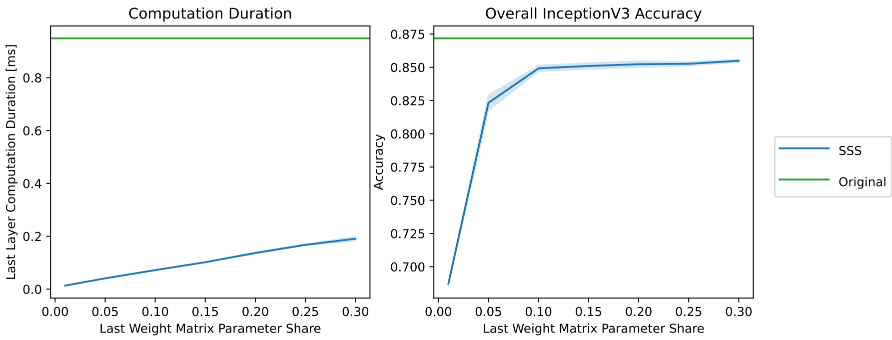

Fig. 2: Evaluating the last layer of the InceptionV3 model requires less time after replacing the weight matrix with an SSS matrix. However, the prediction accuracy of the InceptionV3 model also decreases.

followed by training led to better results than training a similar SSS matrix from scratch. The achieved final accuracy is lower than when using the original weight matrix, whereas the gap between the original accuracy and the accuracy of the model with SSS matrix depends on the model. For some models, the combined approach of approximating and training the SSS weight matrices achieved very good results, yielding a good trade-off between reduction in parameters and reduction of accuracy in these models. These results are depicted in figure 1.

Besides the accuracy of the final model, we are also interested in the duration required for running inference. For that, we compared the time required for evaluating the last layer of the InceptionV3 model (depicted in Figure 2). This speed comparison is implemented in C and executed on a single CPU core (of an Intel Core i-7-8750H CPU with 2.20 GHz). The time required for evaluating the matrix-vector multiplication increases with the number of parameters in the SSS matrix. For all investigated parameter shares the required computation time is significantly lower after replacing the original weight matrix with an SSS matrix.

5 Conclusion

We analyzed the effect of replacing weight matrices in the dense layers of deep CNNs with SSS matrices. The resulting modified layers require significantly less parameters to be stored, and can be evaluated much faster than the original layers. This is due to the structure of the SSS matrix, which facilitates efficient matrix-vector multiplication with subquadratic order of operations.

Our results showed that there is a trade-off between reducing the number of parameters and decrease in prediction accuracy. The performance depends on the number of parameters in the SSS matrix and the model at hand. We conclude that there is a lot of potential in the approach of optimizing trained neural networks by introducing SSS matrices.

References

[1] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[2] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[3] Patrick Dewilde and Alle-Jan Van der Veen. *Time-varying systems and computations*. Springer Science & Business Media, 1998.

[4] Victor Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.

[5] Vikas Sindhwani, Tara N Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 3088–3096, 2015.

[6] Anna T Thomas, Albert Gu, Tri Dao, Atri Rudra, and Christopher Ré. Learning compressed transforms with low displacement rank. *Advances in neural information processing systems*, 2018:9052, 2018.

[7] Liang Zhao, Siyu Liao, Yanzhi Wang, Zhe Li, Jian Tang, and Bo Yuan. Theoretical properties for neural networks with weight matrices of low displacement rank. In *international conference on machine learning*, pages 4082–4090. PMLR, 2017.

[8] Wolfgang Hackbusch. *Hierarchical matrices: algorithms and analysis*, volume 49. Springer, 2015.

[9] Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *International Conference on Learning Representations*, 2020.

[10] Yuwei Fan, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical matrices. *Multiscale Modeling & Simulation*, 17(4):1189–1213, 2019.

[11] Yuwei Fan, Jordi Feliu-Faba, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical nested bases. *Research in the Mathematical Sciences*, 6(2):1–28, 2019.

[12] Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *International Conference on Machine Learning*, pages 1517–1527. PMLR, 2019.

[13] Luc Giffon, Stéphane Ayache, Hachem Kadri, Thierry Artières, and Ronan Sicre. Psm-nets: Compressing neural networks with product of sparse matrices. 2021.

[14] Nir Ailon, Omer Leibovitch, and Vineet Nair. Sparse linear networks with a fixed butterfly structure: theory and practice. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161, pages 1174–1184. PMLR, 2021.

[15] Matthias Kissel, Martin Gottwald, Biljana Gjeroska, Philipp Paukner, and Klaus Diepold. Backpropagation through states: Training neural networks with sequentially semiseparable weight matrices. *Proceedings of the 21st EPIA Conference on Artificial Intelligence*, 2022.

[16] Matthias Kissel, Sven Gronauer, Mathias Korte, Luca Sacchetto, and Klaus Diepold. Exploiting structures in weight matrices for efficient real-time drone control with neural networks. *Proceedings of the 21st EPIA Conference on Artificial Intelligence*, 2022.

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

5.5. Neural Networks comprising Sequentially Semiseparable Matrices with one dimensional State Variable are Universal Approximators

Authors: Matthias Kissel and Klaus Diepold
Journal: ECML 2023 Proceedings (Springer Communications in Computer and Information Science)
Publisher, publication year: Springer, 2023
Core publication: Yes
Bibliography Entry: [49]
Comment: Acceptance letter for this publication can be found in Appendix B. Reproduced with permission from Springer Nature.

Summary

In this paper, we analyze the approximation capabilities of NNs comprising SSS weight matrices. In particular, we investigate SSS matrices with one dimensional state variable. This class of matrices is quite limited in their expressiveness, but it facilitates an efficient matrix-vector multiplication algorithm. Our contribution is to prove that neural networks comprising SSS matrices with one dimensional state variable are universal approximators. With our proof, we show that the same approximation capabilities which have been shown for weight matrices of low displacement rank also apply for SSS weight matrices. I use this result in this thesis as a basis to analyze the benefits of using SSS weight matrices in NNs. In the proof, it is shown that these NNs can approximate any function with arbitrary accuracy. With this knowledge, experiments can be conducted to investigate the trade-off between approximation accuracy and number of parameters in the NN.

Own Contributions

- Literature Research for existing proofs and similar approaches
- Development of the proof presented in the paper and formulation of the Universal Approximation Theorem in collaboration with Klaus Diepold
- Discussion of the presented proof and its implications

Neural Networks comprising Sequentially Semiseparable Matrices with one dimensional State Variable are Universal Approximators

Matthias Kissel[1] and Klaus Diepold[1]

[1]Technical University of Munich, Arcisstr. 21, 80333 Munich, Germany
matthias.kissel@tum.de

Abstract. One approach towards handling the large resource requirements of modern neural networks is to use structured weight matrices. In this paper, we analyze the approximation capabilities of such neural networks. In particular, we investigate sequentially semiseparable (SSS) matrices with one dimensional state variable. This class of matrices is quite limited in their expressiveness, but it facilitates an efficient matrix-vector multiplication algorithm. Our contribution is to prove that neural networks comprising SSS matrices with one dimensional state variable are universal approximators. With our proof, we show that the same approximation capabilities which have been shown for weight matrices of low displacement rank also apply for SSS weight matrices.

Keywords: Matrix Structures · Efficient Inference · Sequentially Semiseparable Matrices.

1 Motivation

Modern neural networks achieve remarkable results in several domains. This is based, among other things, on the fact that networks are becoming ever larger and deeper. State-of-the-art networks often comprise millions of parameters [15, 26], requiring large computational resources for training and inference. Some applications even require specialized hardware for using these networks [22].

One approach to deal with this increasing resource consumption is to use structured weight matrices.

Definition 1. *A matrix $A \in \mathbb{R}^{m \times n}$ is called structured, if it is defined by less than $\mathcal{O}(mn)$ parameters.*

In contrast to sparse matrices, structured matrices don't need to contain zeros. Instead, a structured matrix can be dense and is defined by the relationship of few parameters. Besides the apparent memory savings, there are efficient algorithms for some classes of structured matrices that can save computational resources for performing various linear algebra operations.

There are many types of structured matrices. Two prominent examples are hierarchical matrices [2] and matrices of low displacement rank [20]. In this paper, we focus on another structure class that occurs when describing time-varying systems using state-space methods [9]: SSS matrices. The number of parameters defining an SSS matrix is inter alia determined by the dimension of the state variable of the described system. Every matrix can be represented as SSS matrix if the state dimension is large enough. However, matrices defined with low dimensional state variable are of particular interest. For these matrices, there exist an efficient matrix-vector multiplication algorithm [13, 3]. This means that memory as well as computational resources can be saved when the matrix is represented as SSS matrix. Matrix-vector multiplications play a major role for the computational cost required for inference with neural networks [28, 19]. Therefore, using SSS matrices in neural networks can significantly reduce the computational demands of neural networks.

Typically, it is not evident which matrix structure type is best suited for a given problem. This is, because there is not one single structure which outperforms all others when being used in neural networks. Moreover, there is a trade-off between reduction in parameters, inference time, and prediction accuracy of the resulting model. Therefore, it is important to have a repertoire of possible matrix structures, which can be used in neural networks. By that, different structure types can be tested for the problem at hand. By focusing on SSS matrices in this paper, we give the theoretical foundation needed to add them to the repertoire of matrix structures, which can be used in neural networks.

Many theoretical insights in the field of neural networks build on the universal approximation theorem, proven by Cybenko in [5]. His theorem states that neural networks with sigmoidal activation function can be used to approximate any function to a desired accuracy. In [29], Zhao et al. show that this theorem also holds for neural networks comprising weight matrices of low displacement rank. Based on these results, our main contribution is to prove that Cybenko's [5] universal approximation theorem also holds for neural networks comprising SSS matrices with one dimensional state variable. By that, we show that the same approximation capabilities, which have been shown for neural networks comprising matrices of low displacement rank, also hold for neural networks comprising SSS matrices.

The rest of this paper is organized as follows. We first give an overview over previous work using structured matrices in neural networks. Subsequently, we define sequentially semiseparable matrices and explain how they can be used in neural networks. Our main contribution is given in Section 4, in which we show that the universal approximation holds for neural networks comprising sequentially semiseparable matrices with one dimensional state variable. Finally, we summarize our findings and draw a conclusion.

2 Literature Review

The research about semiseparable matrices dates back until 1937 [12, 25]. This class of matrices has some interesting properties, and there exist efficient algorithms for several applications. For example, the inverse of a generator representable plus diagonal semiseparable matrix can be computed in an efficient way. Vandebril et al. [25] give a comprehensive overview over the results achieved with semiseparable matrices.

A special member of the class of semiseparable matrices are SSS matrices [9], which occur when describing time-varying systems using a state-space representation. We define SSS matrices formally in Section 3. Depending on the properties of the SSS matrix (which refers foremost to the dimension of the state variable), SSS matrices can be efficiently multiplied with vectors. This makes them particularly interesting for use in neural networks, since a large part of the computational costs for the use of neural networks is spent on matrix-vector multiplications. This is why SSS matrices have been used for example in the domain of neural drone control [18], or for approximating large matrices arising in deep convolutional networks [16]. Moreover, Kissel et al. [17] proposed the *backpropagation through states* algorithm, which can be used for training neural networks with SSS weight matrices.

Besides SSS matrices, there are many other types of structured matrices. Some of them have been used in neural networks. For example, Fan et al. [11, 10] used hierarchical weight matrices in neural networks, resulting in a multiscale structure. Especially for products of sparse matrices, there have been promising results recently. A product of sparse matrices is in general not sparse. It has been shown that many dense matrices can be well approximated with products of sparse matrices [7, 8, 6], showing promising results when applied to neural networks [7, 1, 27].

The most popular structure class used in neural networks are matrices of low displacement rank. This class includes well known matrix types like Toeplitz, Hankel, Vandermonde, and Cauchy matrices. Even convolutional neural networks can be described as standard neural network with weight matrices of low displacement rank (using sparse Toeplitz matrices). Sindhwani et al. [23] proposed to use Toeplitz-like matrices, which can be trained end-to-end with the rest of the network. Moreover, Thomas et al. [24] trained displacements as well as operator matrices end-to-end as part of the neural network training procedure. Zhao et al. [29] contributed theoretical results regarding neural networks with weight matrices of low displacement rank. They showed that these networks are universal approximators.

The topics of this paper also touch the concepts of structured sparsity learning [21, 14]. However, we do not focus on sparse matrices. The structured matrices we consider are usually dense and thus do not contain zeros. This distinguishes our approach from structured sparsity learning, which aims to identify zero entries in a matrix that have some structural relationship to each other. However, it is possible that the concepts used to describe the structure in both approaches do overlap with each other.

3 Sequentially Semiseparable Matrices

In the following, we define SSS matrices based on the matrix-vector product $y = Tu$, where $y \in \mathbb{R}^m$ is the resulting vector, $T \in \mathbb{R}^{m \times n}$ is the SSS matrix, and $u \in \mathbb{R}^n$ is the input vector. Analogous to the results from Zhao et al. [29], we consider square matrices in this paper (i.e. $m = n$). This is, however, not a general limitation for SSS matrices, since they are defined for arbitrary matrix shapes.

SSS matrices occur when describing time-varying systems using a state-space representation. In time-varying system theory, T is called the Toeplitz operator, u are the system inputs, and y are the system outputs. The Toeplitz operator describes the time-varying input-output behavior of the system. That is, for each timestep $k = 1, \ldots, p$, the outputs of this timestep y_k for a causal system are computed based on the inputs u_k and the state of the system at this timestep x_k:

$$y_k = C_k x_k + D_k u_k \tag{1}$$

with

$$x_{k+1} = A_k x_k + B_k u_k. \tag{2}$$

A_k, B_k, C_k and D_k for $k = 1, \ldots, p$ are matrices describing the system behavior. The Toeplitz operator corresponding to the system has a particular structure

$$T = \begin{bmatrix} D_1 & 0 & 0 & 0 \\ C_2 B_1 & D_2 & 0 & 0 \\ C_3 A_2 B_1 & C_3 B_2 & D_3 & 0 \\ C_4 A_3 A_2 B_1 & C_4 A_3 B_2 & C_4 B_3 & D_4 \end{bmatrix}. \tag{3}$$

Since we do not investigate physical systems, we call the number of timesteps p computation stages throughout the paper.

In the following, we restrict the dimensions of inputs, outputs and the state variable to one at each computation stage. This limits the expressiveness of the class considerably. As mentioned before, SSS matrices with arbitrary state variable dimension can represent any matrix. With the limitation to one dimensional input, output, and state variable, this expressiveness is lost, and some matrices can no longer be represented. Moreover, we restrict the SSS matrix to be a lower diagonal matrix (which corresponds to a causal Toeplitz operator). This further restricts the expressiveness of the structure class. However, since the class of lower diagonal SSS matrices is contained in the general class of SSS matrices, a proof for lower diagonal SSS matrices directly also applies to general SSS matrices.

With the aforementioned limitations, we can define SSS matrices.

Definition 2. *A lower triangular SSS matrix $T \in \mathbb{R}^{n \times n}$ with one dimensional input, output and state variable at each computation stage, is defined as*

$$T = D + C(I - ZA)^{-1} ZB, \tag{4}$$

where D, C, A, and B are diagonal matrices

$$A = \begin{pmatrix} a_1 & & & 0 \\ & a_2 & & \\ & & \ddots & \\ 0 & & & a_p \end{pmatrix} \tag{5}$$

(other matrices respectively), and Z is a down-shift matrix defined as

$$Z = \begin{pmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & 0 \\ 0 & & 1 & 0 \end{pmatrix}. \tag{6}$$

Note that in the general case, the A, B, C and D matrices are block diagonal matrices. However, since we consider the case that the inputs, outputs and states are one dimensional, the entries on the diagonals of the A, B, C and D matrices result as scalars.

Matrices of the form given in Definition 2 can efficiently be multiplied with a vector using the representation given in Equation 1 and 2. Here, the index k on matrices refers to the k^{th} entry on the diagonal of the matrix. It can be seen that in our case with one dimensional state variable, the matrix-vector multiplication can be computed with $\mathcal{O}(n)$ operations, compared to $\mathcal{O}(n^2)$ operations required by the standard algorithm [13, 3].

We are interested in using matrices as defined in Definition 2 in neural networks. For that, analogously to the approach from Zhao et al. [29], we stack r SSS matrices, and use the resulting matrix as weight matrix in a single layer feed-forward neural network with sigmoidal activation function. The resulting network function is given by

$$N(u) = \sum_{j=1}^{rn} \alpha_j \sigma(w_j^T u + \theta_j). \tag{7}$$

Here, α_j are weighing factors for the outputs at each neuron j and θ_j is the bias of the neuron. The overall weight matrix is defined as

$$W = \begin{bmatrix} T_1 \dots T_r \end{bmatrix}, \tag{8}$$

where $T_1, \dots T_r$ are SSS matrices as defined in Definition 2 and w_j denotes the j^{th} column of W.

4 Universal Approximation Theorem

Cybenko [5] proved that single hidden layered neural networks with sigmoidal activation functions are universal approximators. In his proof, he showed that assuming that the set of functions S represented by a neural network is not dense in the space of continuous functions $C(I_n)$ on the n-dimensional unit cube I_n ($[0,1]^n$) results in a contradiction. For that, he used the Hahn-Banach theorem to show that following his assumption that S is not dense in $C(I_n)$, there must be a linear functional L on $C(I_n)$ with the property that $L \neq 0$, but $L(R) = L(S) = 0$ (with R being the closure of S). This then leads to the contradiction, since the discriminatory function $\sigma(y^T x + \theta)$ is in R for all y and θ. Therefore, the subspace S must be dense in $C(I_n)$.

In the following, we show that the universal approximation theorem formulated by Cybenko also applies to networks comprising SSS matrices with one dimensional state variable, as defined in Equation 7. For that, we show that the two requirements on which the universal approximation theorem for standard feedforward neural networks is based, do also apply for our networks: First, we show that the set of functions of the form $N(u)$ defined in Equation 7 (P in the following) is a linear subspace of $C(I_n)$. Second, we show that all functions of the form $\sigma(y^T x + \theta)$ are contained in P. Our approach is based on Cybenko's work, and also follows the approach from Zhao et al. [29], who showed that neural networks with weight matrices of low displacement rank are universal approximators.

Lemma 1. *The set of functions P of the form $N(u)$ as defined in Equation 7 is a linear subspace of $C(I_n)$.*

Proof. We look at the function $N(u)$ defined in Equation 7. By setting

$$\tilde{\alpha}_j = \beta \alpha_j \quad \forall j, \tag{9}$$

we have

$$\begin{aligned} \forall \beta \in \mathbb{R} : \forall N(u) \in P : \exists \tilde{N}(u) \in P : \\ \tilde{N}(u) = \beta N(u). \end{aligned} \tag{10}$$

With

$$\alpha^{(V)} = \left[\alpha^{(H)} \; \alpha^{(G)} \right] \tag{11}$$

(where $\alpha^{(V)}$ denotes the weighing factors of $V(u)$, other variables respectively),

$$W^{(V)} = \left[W^{(H)} \; W^{(G)} \right], \tag{12}$$

and

$$\theta^{(V)} = \left[\theta^{(H)} \; \theta^{(G)} \right], \tag{13}$$

we have

$$\begin{aligned} \forall H(u), G(u) \in P : \\ V(u) = H(u) + G(u) \in P. \end{aligned} \tag{14}$$

Combining the results in Equation 10 and equation 14, it directly follows that

$$\begin{aligned} \forall H(u), G(u) \in P, \kappa, \gamma \in \mathbb{R} : \\ \kappa H(u) + \gamma G(u) \in P. \end{aligned} \tag{15}$$

We now show that the *representation property* [29] is fulfilled by structured matrices as defined in Definition 2. The representation property is fulfilled, if for any vector $v \in \mathbb{R}^n$, there exist a matrix T such that $v \in \mathbb{R}^n$ is a column of T.

Lemma 2 (Representation Property of SSS matrices with one dimensional state variable).

$$\forall y \in \mathbb{R}^n : \exists T = \begin{bmatrix} t_1 \dots t_n \end{bmatrix} \tag{16}$$

with T of the form as defined in Definition 2 and $t_1 = y$.

Proof. We need to show that it is always possible to have

$$T = D + C(I - ZA)^{-1}ZB = \begin{bmatrix} y * \dots * \end{bmatrix} \tag{17}$$

Since the state variable is one dimensional, A, B, C and D are diagonal matrices. In the following, we refer to the i^{th} column of a matrix A by $A^{(i)}$. ZB is a diagonal matrix shifted down by one entry, in particular

$$ZB^{(1)} = \begin{bmatrix} 0 \; b_1 \; 0 \dots 0 \end{bmatrix}^T . \tag{18}$$

Therefore we have

$$C(I - ZA)^{-1}ZB^{(1)} = b_1(C(I - ZA)^{-1})^{(2)}. \tag{19}$$

This can be seen by looking at the product $G(ZB^{(1)})$ with $G = C(I - ZA)^{-1}$

$$\begin{pmatrix} G_{1,1} & G_{1,2} & \dots & G_{1,n} \\ G_{2,1} & G_{2,2} & \dots & G_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ G_{n,1} & G_{n,2} & \dots & G_{n,n} \end{pmatrix} \begin{pmatrix} 0 \\ b_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} . \tag{20}$$

As $(I - ZA)$ is a bidiagonal matrix, the entries of $(I - ZA)^{-1}$ can be computed using the Neumann expansion [9], and are given by [4]

$$((I - ZA)^{-1})_{i,j} = \begin{cases} 0 & for \quad i < j \\ 1 & for \quad i = j \\ \prod_{f=j}^{i-1} a_f & for \quad i > j \end{cases} , \tag{21}$$

where a_f denotes the f^{th} element on the diagonal of A. Note that we switched the indices in the original formula from Chatterjee as we are considering a *lower*-triangular bidiagonal matrix - using the fact that

$$(I - ZA)^{-1} = (((I - ZA)^T)^{-1})^T. \tag{22}$$

Now it can be seen that

$$D^{(1)} + b_1(C(I-ZA)^{-1})^{(2)} = \begin{pmatrix} d_1 \\ b_1 c_2 \\ b_1 c_3 \prod_{f=2}^{2} a_f \\ \vdots \\ b_1 c_n \prod_{f=2}^{n-1} a_f \end{pmatrix} \tag{23}$$

Therefore, if we set $a_k = b_k = 1$ for all k, $d_1 = y_1$, $d_k = 0$ for all $k > 1$ and $c_k = y_k$ for all k we have $t_1 = y$.

Based on lemma 2, we can now show that any function of the form

$$f(x) = \sigma(y^T x + \theta) \tag{24}$$

can be represented with a neural network as defined in Equation 7, where the number of SSS matrices in the network is limited to one (i.e. $r = 1$).

Corollary 1.

$$\begin{aligned} &\forall y, \theta : \exists T, \tilde{\theta}, \alpha : \\ &\sum_{j=1}^{n} \alpha_j \sigma(t_j^T x + \tilde{\theta}_j) = \sigma(y^T x + \theta), \end{aligned} \tag{25}$$

with $T = \begin{bmatrix} t_1 & \dots & t_n \end{bmatrix}$ *and* T *is of the form defined in Definition 2.*

Proof. According to lemma 2, we can chose T such that $t_1 = y$. Moreover, we set $\alpha_1 = 1$ and $\alpha_j = 0$ for all $j \neq 1$ as well as $\tilde{\theta}_j = \theta$ for all j. This results in

$$\begin{aligned} \sum_{j=1}^{n} \alpha_j \sigma(t_j^T x + \tilde{\theta}_j) &= \sigma(t_1^T x + \theta) \\ &= \sigma(y^T x + \theta). \end{aligned} \tag{26}$$

Using lemma 1 and corollary 1, it directly follows that Cybenko's theorem [5] also applies for neural networks as defined in Equation 7.

Theorem 1 (Universal Approximation Theorem for Neural Networks comprising SSS matrices with one dimensional state variable). *Let* σ *be any continuous discriminatory function. Then functions of the form given in Equation 7 are dense in* $C(I_n)$*. In other words, given any* $f \in C(I_n)$ *and* $\epsilon > 0$*, there is a function* $N(u) \in P$ *for which*

$$|N(u) - f(u)| < \epsilon \quad \forall u \in I_n. \tag{27}$$

Proof. Based on Cybenko's universal approximation theorem, this follows directly from Lemma 1 and corollary 1.

5 Discussion

It is evident that neural networks with *arbitrary* SSS weight matrices are universal approximators. This is, because all matrices can be represented exactly as SSS matrices, if the state dimension is large enough. In contrast, it is not straightforward that neural networks comprising SSS matrices with one dimensional state variable are universal approximators. Our results show that neural networks with SSS weight matrices have the same approximation capabilities as neural networks with weight matrices of low displacement rank [29].

However, these results do not directly lead to neural networks with fewer parameters in practice. This is due to two reasons. First, the proof is based on the fact, that there can be multiple SSS matrices in the neural network. This is in line with previous results for matrices of low displacement rank and the standard universal approximation theorem. For practical applications, however, we are more interested in finding structured weight matrices, which perform *sufficiently well* (in contrast to perfectly represent a desired mapping). There is a trade-off between matrices that solve the problem more accurately and matrices for which there are more efficient algorithms for computing the matrix-vector product.

The second reason is that although we proved that neural networks comprising SSS matrices with one dimensional state variable are universal approximators, we did not present an algorithm to find such networks. The recently introduced *backpropagation through states* algorithm [17] can be used to train neural networks with SSS weight matrices. However, it does not provide any guarantees regarding the approximation error. Thus, an algorithm that finds the best structured neural network with guarantees is still lacking.

Nevertheless, it is an important result that neural networks with one dimensional state variable are universal approximators. This provides a framework for finding practically applicable algorithms and structures that will lead to more efficient neural networks. Finding these algorithms is still an ongoing research topic.

6 Conclusion

We showed that the universal approximation theorem holds for neural networks comprising SSS weight matrices with one dimensional state variable. Thus, we have shown that SSS matrices in neural networks have the same approximation capabilities as matrices of low displacement rank. Our results prove that any function can be learned by a neural network with SSS matrices. However, our result does not include an upper bound on the number of parameters needed in practice to accurately approximate a function to a desired degree.

References

1. Ailon, N., Leibovitch, O., Nair, V.: Sparse linear networks with a fixed butterfly structure: theory and practice. In: Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence. vol. 161, pp. 1174–1184. PMLR (2021)
2. Börm, S., Grasedyck, L., Hackbusch, W.: Hierarchical matrices. Lecture notes **21**, 2003 (2003)
3. Chandrasekaran, S., Dewilde, P., Gu, M., Pals, T., Sun, X., van der Veen, A.J., White, D.: Some fast algorithms for sequentially semiseparable representations. SIAM Journal on Matrix Analysis and Applications **27**(2), 341–364 (2005)
4. Chatterjee, G.: Negative integral powers of a bidiagonal matrix. Mathematics of Computation **28**(127), 713–714 (1974)
5. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems **2**(4), 303–314 (1989)
6. Dao, T., Gu, A., Eichhorn, M., Rudra, A., Ré, C.: Learning fast algorithms for linear transforms using butterfly factorizations. In: International Conference on Machine Learning. pp. 1517–1527. PMLR (2019)
7. Dao, T., Sohoni, N., Gu, A., Eichhorn, M., Blonder, A., Leszczynski, M., Rudra, A., Ré, C.: Kaleidoscope: An efficient, learnable representation for all structured linear maps. In: International Conference on Learning Representations (2020)
8. De Sa, C., Cu, A., Puttagunta, R., Ré, C., Rudra, A.: A two-pronged progress in structured dense matrix vector multiplication. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1060–1079. SIAM (2018)
9. Dewilde, P., Van der Veen, A.J.: Time-varying systems and computations. Springer Science & Business Media (1998)
10. Fan, Y., Feliu-Faba, J., Lin, L., Ying, L., Zepeda-Núnez, L.: A multiscale neural network based on hierarchical nested bases. Research in the Mathematical Sciences **6**(2), 1–28 (2019)
11. Fan, Y., Lin, L., Ying, L., Zepeda-Núnez, L.: A multiscale neural network based on hierarchical matrices. Multiscale Modeling & Simulation **17**(4), 1189–1213 (2019)
12. Gantmakher, F., Krein, M.: Sur les matrices completement non négatives et oscillatoires. Compositio mathematica **4**, 445–476 (1937)
13. Golub, G.H., Van Loan, C.F.: Matrix computations. JHU press (2013)
14. Gui, J., Sun, Z., Ji, S., Tao, D., Tan, T.: Feature selection based on structured sparsity: A comprehensive study. IEEE transactions on neural networks and learning systems **28**(7), 1490–1507 (2016)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
16. Kissel, M., Diepold, K.: Deep convolutional neural networks with sequentially semiseparable weight matrices. ESANN 2022 Proceedings (2022)
17. Kissel, M., Gottwald, M., Gjeroska, B., Paukner, P., Diepold, K.: Backpropagation through states: Training neural networks with sequentially semiseparable weight matrices. Proceedings of the 21st EPIA Conference on Artificial Intelligence (2022)
18. Kissel, M., Gronauer, S., Korte, M., Sacchetto, L., Diepold, K.: Exploiting structures in weight matrices for efficient real-time drone control with neural networks. Proceedings of the 21st EPIA Conference on Artificial Intelligence (2022)
19. Moczulski, M., Denil, M., Appleyard, J., de Freitas, N.: Acdc: A structured efficient linear layer. arXiv preprint arXiv:1511.05946 (2015)

20. Pan, V.: Structured matrices and polynomials: unified superfast algorithms. Springer Science & Business Media (2001)
21. Qiao, L.b., Zhang, B.f., Su, J.s., Lu, X.c.: A systematic review of structured sparse learning. Frontiers of Information Technology & Electronic Engineering **18**, 445–463 (2017)
22. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. nature **550**(7676), 354–359 (2017)
23. Sindhwani, V., Sainath, T.N., Kumar, S.: Structured transforms for small-footprint deep learning. In: Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2. pp. 3088–3096 (2015)
24. Thomas, A.T., Gu, A., Dao, T., Rudra, A., Ré, C.: Learning compressed transforms with low displacement rank. Advances in neural information processing systems **2018**, 9052 (2018)
25. Vandebril, R., Van Barel, M., Golub, G., Mastronardi, N.: A bibliography on semiseparable matrices. Calcolo **42**(3), 249–270 (2005)
26. Xie, D., Xiong, J., Pu, S.: All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6176–6185 (2017)
27. Xu, Z., Li, Y., Cheng, X.: Butterfly-net2: Simplified butterfly-net and fourier transform initialization. In: Mathematical and Scientific Machine Learning. pp. 431–450. PMLR (2020)
28. Yang, Z., Moczulski, M., Denil, M., De Freitas, N., Smola, A., Song, L., Wang, Z.: Deep fried convnets. In: Proceedings of the IEEE international conference on computer vision. pp. 1476–1483 (2015)
29. Zhao, L., Liao, S., Wang, Y., Li, Z., Tang, J., Yuan, B.: Theoretical properties for neural networks with weight matrices of low displacement rank. In: international conference on machine learning. pp. 4082–4090. PMLR (2017)

6. Methods

6.1. Experimental Setting

I investigate the effect of using structured weight matrices in NNs by analyzing the outcomes of several experiments. In these experiments, NNs are trained to fit several benchmark datasets, and I am interested in the accuracy after training. All investigated experiments have the same experimental setting, which is depicted in Figure 6.1. First, the weight matrices of the NN are either initialized randomly (in case of training the NN from scratch), or the weights are taken from a pretrained model (in the fine-tuning case). The NN contains at least one layer with a structured weight matrix, for example the last fully-connected layer of a deep vision model. Second, the NN is trained in order to fit a specific dataset. In my analysis, I examine the training results regarding different datasets ranging from computer vision problems to standard regression tasks. The training is conducted by splitting the data into a training, validation and a test set. The gradients for optimizing the model parameters are computed based on the training set, while the validation set is used to compare results of different training runs used to determine the hyperparameters. Possible hyperparameters are the number of epochs the model is trained, or the number of neurons in a layer. After training the model based on the training and validation sets, the model is evaluated on the test set.

In our papers, we presented experimental results for different datasets, NN architectures, as well as different types of structured matrices. I compare the performance of models used in these experiments in order to answer the research questions posed in Section 4. In the following, I summarize the experimental setting of each of our four papers:

- *Deep Convolutional Neural Networks with Sequentially Semiseparable Weight Matrices* [48]: In this paper, we conduct experiments using the Imagenet dataset [68] and pretrained PyTorch [65] vision models. We replace the weight matrix in the last layer of the model with an SSS matrix. Different approaches are investigated for finding the SSS weight matrices used for replacement: approximating a trained weight matrix, training the SSS weight matrix from scratch, and the combined approach of approximation with subsequent fine-tuning. The prediction performance of the resulting models are compared to each other as well as to the prediction performance of standard NNs.

- *Structured Matrices and their Application in Neural Networks: A Survey* [50]: Similar to the experiments in [48], in this paper, we replace the weight matrix of

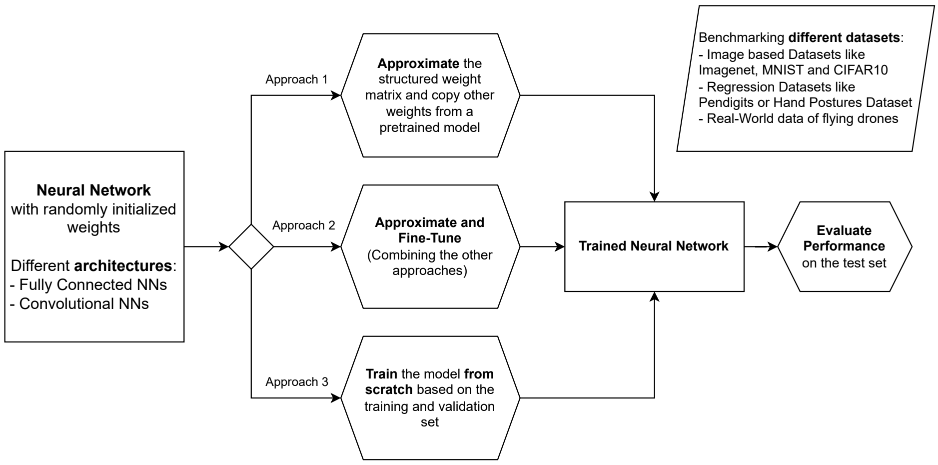

Figure 6.1: Overview over the setting of the analyzed experiments. The network is either a CNN with structured weight matrix in the last layer, or a fully connected NN with structured weight matrices. There are different approaches investigated for training the NN. Moreover, different benchmark datasets are used for the analysis. The outcomes of the experiments are then compared in terms of prediction accuracy on the test set, which has not been used during the training.

the last layer of pretrained PyTorch [65] deep vision models. For that, we use structured matrices of different types (including SSS matrices). We present two benchmarks. In the first benchmark, we approximate test matrices (including weight matrices extracted from the pretrained models) using structured weight matrices of different types. In the second benchmark, the pretrained weight matrices are approximated and subsequently finetuned using gradient-descent based training.

- *Backpropagation Through States: Training Neural Networks with Sequentially Semiseparable Weight Matrices* [51]: In this paper, we compare the prediction performance of trained NNs on several standard benchmark problems. In particular, the prediction performance of standard FFNNs as well as NNs comprising rank 1 weight matrices are compared to the prediction performance of NNs comprising SSS weight matrices. The latter NNs are trained from scratch using the *Backpropagation through states* algorithm, which is introduced in the paper.

- *Exploiting Structures in Weight Matrices for Efficient Real-Time Drone Control with Neural Networks* [52]: This paper investigates NNs with structured matrices used in a real-world setting. For that, weight matrices of NNs capable of controlling quadrotor drones are approximated with structured matrices of different types. Subsequently, we compare the flying capabilities of the resulting NN based controller in simulation as well as on real drones.

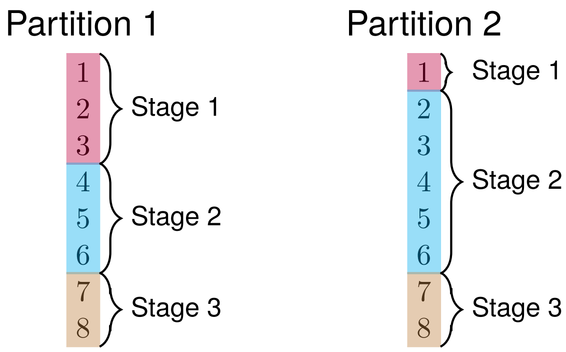

Figure 6.2: Two example partitions for a vector of size 8 (which could either be the outputs before applying the activation function or the inputs to a layer in a NN). Both partitions are suitable and would describe different systems in the real-world. In my experiments, I favor partition 1 over partition 2, because the dimensions of the stages are distributed more evenly in this setting.

In the following sections, I describe the methods used in the mentioned papers in order to bring the SSS structure into the weight matrices of NNs.

6.2. Weight Matrix Partitions

In a standard application scenario, the SSS matrix at hand describes the behavior of a time-varying system. In such a setting, the matrix typically represents the transfer operator of a physical system. Therefore, the input and output dimensions are fixed by the physical conditions. An example of this is a robot that interacts with its environment. At each time step, it is given how many sensor values are available to the robot, and how many actuator control outputs the robot can set. This number determines the u_k and a_k dimensions for all computation stages $k = 1, \ldots, p$ as presented in Section 3.1.

Weight matrices, however, do not describe a physical system, but the relationship between neurons in different layers of a NN. This results in the fact that the partition of a weight matrix with respect to input and output dimensions is not predefined. Specifically, this means that the vector inputs and outputs of the network can be partitioned in various ways. The partition groups the neurons of a layer, such that their outputs before applying the activation function can be computed together at the same computation stage. Moreover, it also defines the order in which the neuron outputs are computed. For example, in a NN, the outputs of neurons $1-9$ could be computed together in the first computation stage. Subsequently in the scond computation stage, the outputs of the neurons $10-14$ might be computed, and so on. This is exemplarily shown for two different partitions in Figure 6.2.

In order to find a suitable partition, I choose to approximately evenly distribute the input and output dimensions across the computation stages. Hence, for a given weight matrix $W \in \mathbb{R}^{n \times m}$, which is to be approximated with an SSS matrix with p stages, the resulting input dimensions are

$$\dim(u_k) = \begin{cases} floor(\frac{m}{p}) + 1 & \text{for} \quad k \leq m - floor(\frac{m}{p})p, \\ floor(\frac{m}{p}) & \text{else} \end{cases} \tag{6.1}$$

and the output dimensions analogously

$$\dim(a_k) = \begin{cases} floor(\frac{n}{p}) + 1 & \text{for} \quad k \leq n - floor(\frac{n}{p})p, \\ floor(\frac{n}{p}) & \text{else} \end{cases}. \tag{6.2}$$

By using the proposed distribution, the number of inputs and outputs are approximately equal in all computational steps. In particular, there are no outliers with a particularly large number of inputs or outputs. This is important for the analysis of the required computational resources, as described in Section 3.1.

Thus, the proposed method represents a simple heuristic to determine the number of inputs and outputs. There is some potential for improvement here. Heuristics can be designed to find better partitions, taking into account the computational resources required for storing the matrix as well as performing matrix-vector multiplications. For example, a recursive method can be used for this as explored in the Master's thesis from Stephan Nüßlein, which I supervised [62]. However, these techniques are outside the scope of this thesis.

I treat the number of computation stages p as hyper parameter, whereas the optimal number of computation stages depends on the problem at hand. Note that the choice of this hyper parameter affects the resource requirements for computing the matrix-vector product. This is, because if p is very small, the average size of the input and output dimension necessarily increases (which might result in higher computational costs, as explained in Section 3.1). In contrast, if p is big, information might need to be passed through many computation stages. This can result in large state vectors required to preserve the state information (or in information loss, if the size of the state vector is restricted). Therefore, I perform a hyper parameter search in order to determine p. For example, values in the range

$$\left[\frac{\min(n, m)}{10}; \frac{\min(n, m)}{2}\right] \tag{6.3}$$

can be used to find a suitable p.

6.3. Approximation of Weight Matrices

Often a matrix is only *approximately* structured. In our case, this means that we can find an SSS matrix that approximately equals a given weight matrix. This can be useful, for example, when a trained NN is given. Then, there is no need to train a new network containing structured matrices. Instead, the standard weight matrices can be replaced by approximated SSS matrices. We analyzed one way to approximate a given weight matrix by an SSS matrix in our papers [48, 52]. In the following, I recapitulate this approach briefly.

Starting from a given weight matrix W, the aim is to find an SSS matrix $\hat{W}$ by minimizing

$$\min_{\hat{W}} ||W - \hat{W}||. \tag{6.4}$$

This can be done using a model order reduction method [21]. As first step, we need to determine a partition of the matrix as explained in the last section (i.e., the dimensions for the inputs, outputs and state variables are to be defined). Subsequently, we cut out the Hankel matrices $\mathcal{H}_k$ from W, which we use to determine a state-space realization Σ for W. Note that in the following, I describe the procedure for determining the causal variables of the realization. The anti-causal variables can be determined analogously.

In order to find a state-space realization, the Hankel matrices are decomposed into observability and reachibility matrices (O_k and C_k respectively). I focus on balanced state-space realization in this thesis, which means that the reachability Gramians $C_k C_k^T$ of the resulting realization equal the observability Gramians $O_k^T O_k$

$$C_k C_k^T = \Theta = O_k^T O_k. \tag{6.5}$$

In order to obtain a balanced realization, the Hankel matrices $\mathcal{H}_k$ are decomposed into observability and reachability matrices in a *balanced* way. This can be done using the Singular Value Decomposition (SVD)

$$\mathcal{H}_k = U_k S_k V_k^T. \tag{6.6}$$

We are not interested in finding an *exact* realization for W, but a realization $\hat{\Sigma}$, which is close to Σ, but matches the predefined partition. In particular, the size of the state variables d_k, which are defined by the ranks of the Hankel matrices

$$d_k = \mathrm{rank}(\mathcal{H}_k), \tag{6.7}$$

is limited. I use a model order reduction method (called balanced model reduction in the time-invariant case) [21] to ensure that the ranks of the Hankel matrices are small enough. For that, I use the d_k largest singular values from S_k and cut out the others

$$\tilde{S}_k = \begin{pmatrix} \sigma_{k,1} & & & & & \\ & \ddots & & & & \\ & & \sigma_{k,d_k} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix}, \tag{6.8}$$

where $\sigma_{k,j}$ is the j^{th} singular value in S_k. The remaining singular values are then evenly distributed in order to obtain the observability and reachibility matrices of the balanced realization

$$\hat{O}_k = U_k\sqrt{\hat{S}_k} \tag{6.9}$$

and

$$\hat{C}_k = \sqrt{\hat{S}_k}V_k^T\,. \tag{6.10}$$

Subsequently A_k, B_k, C_k and D_k can be determined using O_k and C_k [21], whereas E_k, F_k and G_k can be determined using the reachability and observability matrices of the Hankel matrices corresponding to the anti-causal part of the weight matrix. D_k can be read directly from the diagonal of the weight matrix (taking into account the respective number of inputs and outputs). Moreover, B_k and C_k can be read of the reachability and observability matrices, respectively. Here, B_k is defined by the first columns of the reachibility matrix C_k, and C_k is defined by the first rows of the observability matrix O_k. The number of rows or columns is defined by the corresponding input or output dimension. The A_k matrices can either be computed using the reachability matrices

$$A_k = \overleftarrow{C}_k C_k^{\dagger} \tag{6.11}$$

or the observability matrices

$$A_k = O_k^{\dagger} O_k\!\uparrow\,. \tag{6.12}$$

Here, $\dagger$ refers to the Moore-Penrose pseudo-inverse, $O_k\!\uparrow$ is the upshifted version of the observability matrix, and $\overleftarrow{C}_k$ is the left shifted version of the controllability matrix. The resulting realization $\hat{\Sigma}$ is defined by A_k, B_k, C_k, D_k, E_k, F_k, and G_k. $\hat{\Sigma}$ corresponds to the weight matrix $\hat{W}$, whereas $\hat{W}$ is an approximated version of W, which is defined according to the given partition. We provide a Python implementation of this approximation method in our TVSCLib[1] GitHub repository.

6.4. Training Neural Networks with Sequentially Semiseparable Matrices

NNs are typically trained using gradient-descent based methods like the backpropagation algorithm (see Section 3.2). However, if NNs with SSS weight matrices are trained using the standard backpropagation algorithm, the structure in the weight matrices most likely vanishes. This is due to the fact that in the training procedure, the gradients are computed with respect to the entries in the weight matrices, without accounting for the structure in the matrix. In order to overcome this problem, we propose a training algorithm called *Backpropagation through states* [51] designed for training NNs with SSS weight matrices. Using our algorithm, it is ensured that the structure in

[1] https://github.com/MatthiasKi/tvsclib

the SSS weight matrices is preserved throughout the training. In the following, I briefly recapitulate the approach described in the paper.

The key idea of the *Backpropagation through states* algorithm is to derive the training loss with respect to the parameters defining the structure, in contrast to derive the loss with respect to the entries in the weight matrices. SSS matrices are defined by the submatrices $A_k, B_k, C_k, D_k, E_k, F_k$, and G_k for $k = 1, \dots, p$ (see Section 3.1). In our approach, the training loss $\mathcal{L}(a, \hat{a})$ based on the prediction of the network $\hat{a}$ and the target output a is derived with respect to these submatrices

$$\frac{\delta \mathcal{L}(a, \hat{a})}{\delta A_k} \tag{6.13}$$

(other matrices analogously). These gradients can be used to update the submatrices

$$A_k^{(f+1)} = A_k^{(f)} - \alpha \frac{\delta \mathcal{L}(a, \hat{a})}{\delta A_k^{(f)}} \text{ for } f = 1, \dots, L, \tag{6.14}$$

where (f) indicates the f^{th} training step and α is the step size used for optimization. By optimizing the submatrices which define the weight matrix, the entries in the weight matrix change without altering the structure of the weight matrix. Particularly, the size of the inputs, outputs and state variables stays the same throughout the training.

The matrices C_k, D_k and G_k contribute only to a single output segment. In particular, they have no influence on the states x_k or $\hat{x}_k$. Therefore, their gradients are given by

$$\frac{\delta \mathcal{L}(a, \hat{a})}{\delta C_k} = \frac{\mathcal{L}(a, \hat{a})}{\delta a_k} x_k^T, \tag{6.15}$$

$$\frac{\delta \mathcal{L}(a, \hat{a})}{\delta D_k} = \frac{\mathcal{L}(a, \hat{a})}{\delta a_k} u_k, \tag{6.16}$$

and

$$\frac{\delta \mathcal{L}(a, \hat{a})}{\delta G_k} = \frac{\mathcal{L}(a, \hat{a})}{\delta a_k} \hat{x}_{k+1}^T. \tag{6.17}$$

In contrast, A_k, B_k, E_k and F_k do influence the states x_k and $\hat{x}_k$. By altering the states, these matrices have an influence on past or future outputs (depending if they alter the causal or the anti-causal state). This effect needs to be considered when computing the gradients, resulting in

$$\frac{\delta \mathcal{L}(a, \hat{a})}{\delta A_k} = \sum_{s=k+1}^{p} \frac{\delta \mathcal{L}(a, \hat{a})}{\delta \hat{a}_s} \frac{\delta(\tilde{C}(s,k) A_k x_k)}{\delta A_k} \tag{6.18}$$

$$= \sum_{s=k+1}^{p} \frac{\delta \mathcal{L}(a, \hat{a})}{\delta \hat{a}_s} \left(x_k^T \otimes \tilde{C}(s,k)^T \right) \tag{6.19}$$

and

$$\frac{\delta \mathcal{L}(a,\hat{a})}{\delta B_k} = \sum_{s=k+1}^{p} \frac{\delta \mathcal{L}(a,\hat{a})}{\delta \hat{a}_s} \frac{\delta(\tilde{C}(s,k) B_k u_k)}{\delta B_k} \tag{6.20}$$

$$= \sum_{s=k+1}^{p} \frac{\delta \mathcal{L}(a,\hat{a})}{\delta \hat{a}_s} \left(u_k^T \otimes \tilde{C}(s,k)^T \right), \tag{6.21}$$

where $\otimes$ denotes the Kronecker product and

$$\tilde{C}(s,k) = C_s \prod_{k+1}^{f=s-1} A_f. \tag{6.22}$$

The gradients for the anticausal part (E_k and F_k) can be computed analogously. During these computations, the gradients are propagated through the states, giving the algorithm its name. Note that in modern machine learning frameworks like for example PyTorch [65], these gradients can also be computed using automatic differentiation tools without using analytical formulas. We provide a Python implementation of the *Backpropagation through states* training algorithm in our *StructuredNets* GitHub repository[2].

In the course of setting up an NN, the parameters of the network (thus its weight matrices and biases) are usually initialized randomly. This step is also necessary, if an NN with SSS weight matrices is to be trained from scratch. In our experiments [51], we noticed that initializing the submatrices $A_k, \ldots, G_k$ randomly can lead to numerical instabilities. Empirically, it turned out that approximating a random matrix in order to initialize the submatrices leads to a more stable training procedure. The approach presented in the previous section can be used to perform the approximation. For that, first, a standard matrix is initialized randomly. This can be done using a standard approach like for example Glorot-uniform initialization [29]. During the approximation, the input, output and state dimensions are set in order to match the desired dimensions. It should be noted, that the approximated weight matrix does not equal the randomly initialized matrix in general. This is due to the fact that in the approximation step, state dimensions might be pruned.

[2] https://github.com/MatthiasKi/structurednets

7. Results and Discussion

7.1. Benefits of Structured Weight Matrices in Neural Networks

I first analyze the effect of using SSS matrices as weight matrices in NNs. These networks can represent any function just like standard NNs. This is evident by the fact that any weight matrix can be represented by an SSS matrix if the SSS matrix is parameterized with enough variables. However, we are particularly interested in SSS matrices with few parameters, since we expect to save memory and computational resources when performing operations with these matrices. Interestingly, the Universal Approximation property holds even for NNs with SSS weight matrices with one dimensional state variable [49]. This means that a NN containing only weight matrices composed of SSS matrices with one dimensional state variable can still approximate any function with arbitrary accuracy. Following from this result, these networks have the same approximation capabilities as standard NNs [14], as well as NNs with weight matrices of low displacement rank. For the latter, the Universal Approximation Theorem has been proven by Zhao et al. [87]. However, the proof is based on the assumption that any number of SSS matrices with one-dimensional state variable may be combined in the NN. Therefore, despite the proof of the universal approximation theorem, the question remains open of what advantages the use of SSS weight matrices in NNs brings with respect to generalization capabilities and resource consumption. To answer this question, I look at the results from three of our papers [48, 51, 52].

In the first paper [51], we analyzed the prediction performance of NNs with SSS weight matrix on several standard benchmark problems [2, 18, 26, 54]. The models were trained using *backpropagation through states*, starting from randomly initialized weight matrices. Depending on the hyper parameter setting, the models with SSS weight matrix were able to outperform their standard counterpart in all benchmark problems. In our second paper [48], we replaced the last weight matrix of several deep CNNs [41, 55, 70, 72, 76, 77] pretrained on the Imagenet dataset [17] by SSS matrices. In contrast to the first paper, the modified CNNs did not achieve higher prediction accuracy than the original models. I suspect that this is due to the fact that these models were pretrained with standard weight matrices. The SSS matrices were added to the model *after* pretraining. Therefore, the SSS weight matrix needed to adapt to the rest of the model. Training the whole model with structured matrices from scratch might lead to better results. Regarding our experiments with NNs controlling drones [52], we observed that the standard weight matrices outperformed the SSS weight matrices in terms of their ability to robustly control the drone. We suspect that this is due to the network size used in the experiments. Since SSS matrices typically show their benefits if the matrix dimensions become large. In our experiments, however, at most 30 neurons were used in the hidden layers, thus leading to rather small weight matrices. In contrast, other types of structured matrices like products of sparse matrices [56] or matrices of low displacement rank [63] showed better performance after approximation.

Considering these results, I conclude that for some problems, models with SSS weight matrices are able to outperform their standard counterparts in terms of generalization capabilities. However, this does not hold in general. For NNs trained with standard methods, introducing SSS weight matrices seemed to result in inferior prediction accuracy. Therefore, I conclude that Hypothesis 1.1 is false in general, but can be true depending on the problem at hand.

Besides the accuracy of the final model, we are also interested in the duration required for performing inference. In our paper [51], we concluded that the reduction in computation time depends on the architecture to which the model is deployed. For our experiments performed on the microcontroller of the introductory drone example (see Section 1.2), the time required for computing the matrix-vector product was significantly shorter for SSS matrices with few parameters compared to dense matrices. This is in line with the expectation and has also been shown for other types of structured matrices [52] like products of sparse matrices [56]. As a result, SSS matrices *can* be used to achieve a higher control frequency when controlling drones with NNs. However, we observed that on computing hardware with parallelization capabilities (such as multi-core CPUs), the execution time for computing the product between an SSS matrix and a vector could be higher than for a standard matrix-vector multiplication, even if the total number of operations required to compute the result is lower. This is due to the sequential computation order required for computing the matrix-vector product with an SSS matrix. In the domain of time-varying systems, this reflects the fact that the state of the previous timestep is required to compute the state and output of the current time step (in the causal case). Therefore, this algorithm cannot be parallelized as much as the standard algorithm for calculating the matrix-vector product. We made the observation that the use of SSS matrices can lead to shorter inference times in our second paper as well [48]. Here we measured the time required for inference on a single processor core (i.e., on non-parallelized hardware).

Based on these observations, I conclude that Hypothesis 1.2 is correct. The time needed for propagating information through the NN *can* be decreased by using SSS weight matrices. However, this also depends on the architecture to which the NN is deployed. Significant speedups can only be expected on non-parallelized hardware such as single-core processors, microcontrollers, or embedded hardware.

In summary, there are benefits when using SSS matrices in NNs. Depending on the problem at hand, the prediction accuracy of a model with SSS weight matrices can be higher than its standard counterpart. This is, however, not always the case. Moreover, the time required for inference can be reduced by using SSS weight matrices in NNs. This results in a trade-off between the number of parameters in the SSS matrix and the time required for computing the matrix-vector product.

7.2. Impact of the Training Method Choice on the Test Accuracy

The second research question is about the influence of the choice for the method used to bring the SSS structure into the weight matrices of NNs. In order to analyze this, I compare three different methods for bringing structure into the NN. In the first method, the SSS weight matrix is initialized randomly, followed by gradient-descent based training (using *backpropagation through states* [51]). This *data driven* approach is called *training from scratch* in the following. For the second method, the NN is first trained using any training method (like standard backpropagation). Afterwards, the trained weight matrices are approximated with SSS matrices (i.e., this method is *non data driven*). By that, the weight matrices in the trained network can be replaced with their structured counterparts. The third method combines the first two approaches. First, trained weight matrices are approximated with SSS matrices. Subsequently, the network is trained using backpropagation through states (this step is called *fine-tuning* in the following).

We compared all three approaches applied to deep CNNs [41, 55, 70, 72, 76, 77] used for image classification in our paper [48]. One result of the paper is that the method of replacing the original weight matrix with an approximated SSS matrix led to bad results in all experiments. Specifically, for all investigated models and datasets, the data driven approach of training the SSS weight matrices led to higher test accuracy. These results are supported by observations made in another paper of ours [52]. Here, we replaced weight matrices in a NN used for controlling a drone by structured matrices of different types. Replacing the trained weight matrices with SSS matrices led to bad flying performance most of the time. However, the size of the investigated networks may have had an impact on the results (since the networks were very small). Note that in contrast to these results regarding the use of SSS weight matrices, a good flying performance could be achieved by using other types of structured matrices for approximation. The trade-off between reduction of parameters in the network and resulting flying performance was particularly good for products of sparse matrices [56], which has also been observed in other works [16, 28]. Based on the evidence of both papers, I conclude that Hypothesis 2.1 is true: The data driven approach of optimizing SSS weight matrices leads to better results than approximating trained weight matrices. This is in line with the results for other types of structured matrices given in our survey paper [50]. In the fine-tuning benchmark of the paper, it was shown that for all matrix structure types (including products of sparse matrices [56] and hierarchical matrices [39]), it was possible to improve the prediction accuracy by fine-tuning the approximated weight matrices.

Comparing the approaches of training the SSS weight matrices from scratch against fine-tuning SSS matrices after approximation, the fine-tuning approach led to the best results. Note that both of these methods incorporate a data driven training stage. I assume that the results of the experiments would also apply to other problem domains, since the behavior has been observed for many different NNs and two different subsets of Imagenet classes. Thus, I conclude that Hypothesis 2.2. is also verified.

Since both hypotheses associated with Research Question 2 have been verified, I conclude that the choice of the training method used for introducing SSS matrices into NNs *does* have an impact on the achieved test accuracy. Our experiments indicate that the most promising approach is to approximate trained weight matrices with SSS matrices, followed by fine-tuning using backpropagation through states.

7.3. Impact of the Structure Class Choice on the Test Accuracy

The third research question examines the impact of the structure class chosen to be used in the NN. One of the main characteristics in the selection of suitable matrix structures is the number of parameters needed to describe a matrix. The question arises whether the type of structure is playing a role at all for the prediction accuracy of the overall NN when matrix structures with the same number of parameters are compared to each other.

First, I investigate the effect of the structure class choice when approximating a trained weight matrix. For that, I look at the results in our paper [52], which considers a similar experimental setting as introduced in my introductory example in Section 1.2. Here, we approximated weight matrices of NNs used for controlling drones with different types of structured matrices. Specifically, we compare four approximation approaches based on different matrix structure types, namely low rank approximation based on the SVD, approximating matrices of low displacement rank based on the approach from Thomas et al. [78], approximating SSS matrices using time-varying systems theory [21], and approximating products of sparse matrices based on the approach proposed by Magoarou and Gribonval [56]. Each of the approximated matrices contained (almost) the same number of parameters. The results showed that depending on the chosen structure, the resulting performance of the NNs with approximated weight matrices were very different. This effect was particularly visible when the approximated matrices comprised few parameters. In this case, for example, products of sparse matrices tended to outperform other types of structure. The same conclusion can be drawn when looking at the benchmark results presented in our survey paper [50]. In this benchmark, we approximated several test matrices with structured matrices of different types. We observed that choosing the right structure for approximation depends on the problem at hand. Nevertheless, products of sparse matrices achieved very promising results in most cases. Based on these results I conclude that Hypothesis 3.1 is false. The structure type used for approximating trained weight matrices *does* play a role even if the number of parameters is the same.

A similar effect can be observed when training structured weight matrices in NNs. In our paper [51], we compared the prediction results of NNs with different types of structured weight matrices trained from scratch. In particular, we compared the prediction performance of NNs with rank one weight matrices to NNs with SSS weight matrices. Our results showed that standard NNs outperformed NNs with low rank weight matrices on most benchmarks. In contrast, NNs with SSS weight matrices achieved better prediction results than the standard models on all benchmark datasets. However, in this comparison it was not ensured that the networks comprise the same number of trainable parameters. In another benchmark presented in our survey paper [50], we compared the prediction performance of several pretrained deep CNNs for which the weight matrix of the last layer had been replaced with structured matrices of different types. Specifically, we compared the approaches of using low rank matrices, products of sparse matrices [56], hierarchical matrices [39], and SSS matrices. After replacing the weight matrix with a structured matrix, the layer was trained using gradient descent on the same data which was used for pretraining. The results of this benchmark show that the modified NNs achieved different prediction performance. Indeed, after the gradient-descent based training, the NNs with SSS weight matrices showed the worst prediction accuracy. In contrast, networks with products of sparse weight matrices showed consistently good performance. For some CNNs architectures, this structure type achieved a similar prediction accuracy as the baseline models, even though there were significantly fewer parameters in the last weight matrix. This observation is in line with other research results, which showed that NNs with products of sparse matrices as weight matrices can achieve very good prediction performance with few parameters [16, 28]. From that, I conclude that Hypothesis 3.2 is false. Also when training NNs with structured weight matrices using gradient-descent, the choice of the structure class used in the network has an impact on the achieved test accuracy.

Since both hypotheses of Research Question 3 are falsified, I conclude that the choice of the structure class introduced into the weight matrices of NNs *does* have an impact on the achieved test accuracy. Choosing a suitable structure type seems to depend on the problem at hand. In our experiments with NNs used for controlling drones, for example, products of sparse matrices consistently achieved good results. I suspect that the difference in performance is due to the fact that some structure types can represent more matrices than others. For example, any low rank matrix can also be represented in the framework of SSS matrices. Therefore, it can be expected that the use of SSS matrices in NNs leads to better results (if the matrices are large enough).

8. Conclusion

In this work, I investigate NNs with structured weight matrices. Particularly, the focus lies on SSS weight matrices. I am interested in the effects of using SSS weight matrices in NNs with focus on resource consumption and generalization capabilities, also compared to other structure types. In order to examine these effects, I look at three research questions.

The first question asks about benefits, which can be gained by using SSS weight matrices in NNs. As result of my research, I find that for some problems, NNs with SSS weight matrices can achieve better test prediction accuracy even if they comprise fewer parameters. Moreover, by exploiting the structure of the weight matrices, the time required for inference can be reduced (depending on the hardware architecture being used).

Following the second research question, I next analyze the influence of the training method used to bring the structure into the NN. Here, I compare three methods: approximating given weight matrices, training SSS weight matrices from scratch, or the combined approach of first approximating given weight matrices followed by training (fine-tuning). The combined approach of approximation and training showed to be the most promising one, thus achieving the best prediction results in the conducted experiments. I also observe that data driven training approaches outperformed approximation based methods in most experiments.

My third research question aims at comparing different matrix structures, which can be used in NNs. The goal is to investigate the impact of the structure class choice with respect to the number of trainable parameters in the structured matrix. For both training methods, gradient-descent based training as well as approximation, the experiments show that the choice of the structure class has an impact on the achieved results. This leads to the conclusion that the structure class used in the NN should be selected in order to fit the problem at hand.

Overall, the results show that the use of structured matrices in NNs is promising in terms of required computing and storage resources. This is particularly important for modern network architectures, which often comprise several million parameters. Moreover, the generalization capability of NNs can be increased using structured weight matrices depending on the given problem. However, it is not trivial to choose the right structure for a given problem. Incorporating any structured matrix does not necessarily lead to improvements.

Bibliography

[1] N. Ailon, O. Leibovitch, and V. Nair. "Sparse linear networks with a fixed butterfly structure: theory and practice". In: *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*. Vol. 161. PMLR, 2021, pp. 1174–1184.

[2] F. Alimoglu and E. Alpaydin. "Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition". In: *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN 96*. Citeseer. 1996.

[3] D. Bini and V. Y. Pan. *Polynomial and matrix computations: fundamental algorithms*. Springer Science & Business Media, 2012.

[4] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. "What is the state of neural network pruning?" In: *Proceedings of machine learning and systems* 2 (2020), pp. 129–146.

[5] S. Börm and L. Grasedyck. "Hybrid cross approximation of integral operators". In: *Numerische Mathematik* 101 (2005), pp. 221–249.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[7] J. R. Bunch. "Stability of methods for solving Toeplitz systems of equations". In: *SIAM Journal on Scientific and Statistical Computing* 6(2) (1985), pp. 349–364.

[8] P. Casella and A. Paiva. "Magenta: An architecture for real time automatic composition of background music". In: *International Workshop on Intelligent Virtual Agents*. Springer. 2001, pp. 224–232.

[9] C. Chalmers, P. Fergus, C. A. Curbelo Montanez, S. N. Longmore, and S. A. Wich. "Video analysis for the detection of animals using convolutional neural networks and consumer-grade drones". In: *Journal of Unmanned Vehicle Systems* 9(2) (2021), pp. 112–127.

[10] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White. "Some fast algorithms for sequentially semiseparable representations". In: *SIAM Journal on Matrix Analysis and Applications* 27(2) (2005), pp. 341–364.

[11] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A.-J. van der Veen. "Fast stable solver for sequentially semi-separable linear systems of equations". In: *High Performance Computing—HiPC 2002: 9th International Conference Bangalore, India, December 18–21, 2002 Proceedings*. Springer. 2002, pp. 545–554.

[12] C. Chen, S. Reiz, C. Yu, H.-J. Bungartz, and G. Biros. "Fast Approximation of the Gauss–Newton Hessian Matrix for the Multilayer Perceptron". In: *SIAM Journal on Matrix Analysis and Applications* 42(1) (2021), pp. 165–184.

[13] J. Chen and X. Ran. "Deep learning with edge computing: A review". In: *Proceedings of the IEEE* 107(8) (2019), pp. 1655–1674.

[14] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2(4) (1989), pp. 303–314.

[15] T. Dao, A. Gu, M. Eichhorn, A. Rudra, and C. Ré. "Learning fast algorithms for linear transforms using butterfly factorizations". In: *International conference on machine learning*. PMLR. 2019, pp. 1517–1527.

[16] T. Dao, N. S. Sohoni, A. Gu, M. Eichhorn, A. Blonder, M. Leszczynski, A. Rudra, and C. Ré. "Kaleidoscope: An efficient, learnable representation for all structured linear maps". In: *arXiv preprint arXiv:2012.14966* (2020).

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[18] L. Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29(6) (2012), pp. 141–142.

[19] T. Dettmers and L. Zettlemoyer. "Sparse networks from scratch: Faster training without losing performance". In: *arXiv preprint arXiv:1907.04840* (2019).

[20] P. Dewilde and A.-J. van der Veen. "Inner–outer factorization and the inversion of locally finite systems of equations". In: *Linear Algebra and its Applications* 313(1-3) (2000), pp. 53–100.

[21] P. Dewilde and A.-J. Van der Veen. *Time-varying systems and computations*. Springer Science & Business Media, 1998.

[22] I. S. Duff, R. G. Grimes, and J. G. Lewis. "Sparse matrix test problems". In: *ACM Transactions on Mathematical Software (TOMS)* 15(1) (1989), pp. 1–14.

[23] Y. Fan, J. Feliu-Faba, L. Lin, L. Ying, and L. Zepeda-Núnez. "A multiscale neural network based on hierarchical nested bases". In: *Research in the Mathematical Sciences* 6(2) (2019), pp. 1–28.

[24] Y. Fan, L. Lin, L. Ying, and L. Zepeda-Núnez. "A multiscale neural network based on hierarchical matrices". In: *Multiscale Modeling & Simulation* 17(4) (2019), pp. 1189–1213.

[25] F. Gantmakher and M. Krein. "Sur les matrices complètement non négatives et oscillatoires". In: *Compositio mathematica* 4 (1937), pp. 445–476.

[26] A. Gardner, J. Kanno, C. A. Duncan, and R. Selmic. "Measuring distance between unordered sets of different sizes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 137–143.

[27] P. Gerds and L. Grasedyck. "Solving an elliptic PDE eigenvalue problem via automated multi-level substructuring and hierarchical matrices". In: *Computing and Visualization in Science* 16(6) (2013), pp. 283–302.

[28] L. Giffon, S. Ayache, H. Kadri, T. Artières, and R. Sicre. "PSM-nets: Compressing Neural Networks with Product of Sparse Matrices". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.

[29] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Vol. 9. Proceedings of Machine Learning Research. PMLR, May 2010, pp. 249–256.

[30] G. H. Golub and C. F. Van Loan. *Matrix computations*. JHU press, 2013.

[31] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[32] L. Grasedyck. "Hierarchical singular value decomposition of tensors". In: *SIAM Journal on Matrix Analysis and Applications* 31(4) (2010), pp. 2029–2054.

[33] L. Grasedyck, W. Hackbusch, and B. N. Khoromskij. "Solution of large scale algebraic matrix Riccati equations by use of hierarchical matrices". In: *Computing* 70 (2003), pp. 121–165.

[34] J. L. Greathouse and M. Daga. "Efficient sparse matrix-vector multiplication on GPUs using the CSR storage format". In: *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2014, pp. 769–780.

[35] S. Gronauer, M. Kissel, L. Sacchetto, M. Korte, and K. Diepold. "Using simulation optimization to improve zero-shot policy transfer of quadrotors". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 10170–10176.

[36] F. Gustavson and D. Yun. "Fast algorithms for rational Hermite approximation and solution of Toeplitz systems". In: *IEEE Transactions on Circuits and Systems* 26(9) (1979), pp. 750–755.

[37] W. Hackbusch. *Hierarchical matrices: algorithms and analysis*. Vol. 49. Springer, 2015.

[38] W. Hackbusch. "Survey on the technique of hierarchical matrices". In: *Vietnam Journal of Mathematics* 44 (2016), pp. 71–101.

[39] W. Hackbusch and S. Börm. “Data-sparse approximation by adaptive H2-matrices”. In: *Computing* 69(1) (2002), pp. 1–35.

[40] B. Hassibi and D. Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

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

[42] P. Hill, A. Jain, M. Hill, B. Zamirai, C.-H. Hsu, M. A. Laurenzano, S. Mahlke, L. Tang, and J. Mars. “Deftnn: Addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission”. In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 2017, pp. 786–799.

[43] V. Ithapu. “Decoding the Deep: Exploring class hierarchies of deep representations using multiresolution matrix factorization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 45–54.

[44] Z. Jiao, Y. Zhang, J. Xin, L. Mu, Y. Yi, H. Liu, and D. Liu. “A Deep Learning Based Forest Fire Detection Approach Using UAV and YOLOv3”. In: *2019 1st International Conference on Industrial Artificial Intelligence (IAI)*. 2019, pp. 1–5.

[45] T. Kailath. “A view of three decades of linear filtering theory”. In: *IEEE Transactions on information theory* 20(2) (1974), pp. 146–181.

[46] T. Karras, T. Aila, S. Laine, and J. Lehtinen. “Progressive growing of gans for improved quality, stability, and variation”. In: *arXiv preprint arXiv:1710.10196* (2017).

[47] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[48] M. Kissel and K. Diepold. “Deep Convolutional Neural Networks with Sequentially Semiseparable Weight Matrices”. In: *ESANN 2022 Proceedings (European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning)* (2022).

[49] M. Kissel and K. Diepold. “Neural Networks comprising Sequentially Semiseparable Matrices with one dimensional State Variable are Universal Approximators”. In: *Springer Communications in Computer and Information Science* (2023).

[50] M. Kissel and K. Diepold. “Structured Matrices and their Application in Neural Networks: A Survey”. In: *New Generation Computing* (2023).

[51] M. Kissel, M. Gottwald, B. Gjeroska, P. Paukner, and K. Diepold. “Backpropagation Through States: Training Neural Networks with Sequentially Semiseparable Weight Matrices”. In: *Proceedings of the 21st EPIA Conference on Artificial Intelligence* (2022).

[52] M. Kissel, S. Gronauer, M. Korte, L. Sacchetto, and K. Diepold. “Exploiting Structures in Weight Matrices for Efficient Real-Time Drone Control with Neural Networks”. In: *Proceedings of the 21st EPIA Conference on Artificial Intelligence* (2022).

[53] A. Krizhevsky. “One weird trick for parallelizing convolutional neural networks”. In: *arXiv preprint arXiv:1404.5997* (2014).

[54] A. Krizhevsky and G. Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. (0). Toronto, Ontario: University of Toronto, 2009.

[55] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60(6) (2017), pp. 84–90.

[56] L. Le Magoarou and R. Gribonval. “Flexible multilayer sparse approximations of matrices and applications”. In: *IEEE Journal of Selected Topics in Signal Processing* 10(4) (2016), pp. 688–700.

[57] Y. LeCun, J. Denker, and S. Solla. “Optimal brain damage”. In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[58] W. Lee, S. Kim, Y.-T. Lee, H.-W. Lee, and M. Choi. “Deep neural networks for wild fire detection with unmanned aerial vehicle”. In: *2017 IEEE International Conference on Consumer Electronics (ICCE)*. 2017, pp. 252–253.

[59] Y. Li, X. Cheng, and J. Lu. “Butterfly-Net: Optimal Function Representation Based on Convolutional Neural Networks”. In: *Communications in Computational Physics* 28(5) (2020), pp. 1838–1885.

[60] Y. Liu, S. Jiao, and L.-H. Lim. “LU decomposition and Toeplitz decomposition of a neural network”. In: *arXiv preprint arXiv:2211.13935* (2022).

[61] E. Lygouras, N. Santavas, A. Taitzoglou, K. Tarchanidis, A. Mitropoulos, and A. Gasteratos. “Unsupervised human detection with an embedded vision system on a fully autonomous UAV for search and rescue operations”. In: *Sensors* 19(16) (2019), p. 3542.

[62] S. Nüßlein. “Algorithms for Matrix Approximations with Time Varying Systems”. en. Master’s thesis. Technische Universität München, 2022.

[63] V. Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.

[64] V. Y. Pan. “Solving a polynomial equation: some history and recent progress”. In: *SIAM review* 39(2) (1997), pp. 187–220.

[65] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[66] A. Rivas, P. Chamoso, A. González-Briones, and J. M. Corchado. “Detection of cattle using drones and convolutional neural networks”. In: *Sensors* 18(7) (2018), p. 2048.

[67] D. Rumelhart, G. Hinton, and R. Williams. “Learning representations by back-propagating errors”. In: *nature* 323(6088) (1986), pp. 533–536.

[68] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115(3) (2015), pp. 211–252.

[69] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[70] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[71] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. “Mastering the game of go without human knowledge”. In: *nature* 550(7676) (2017), pp. 354–359.

[72] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[73] V. Sindhwani, T. N. Sainath, and S. Kumar. “Structured transforms for small-footprint deep learning”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*. 2015, pp. 3088–3096.

[74] E. Strubell, A. Ganesh, and A. McCallum. “Energy and Policy Considerations for Deep Learning in NLP”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 3645–3650.

[75] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer. “Efficient processing of deep neural networks: A tutorial and survey”. In: *Proceedings of the IEEE* 105(12) (2017), pp. 2295–2329.

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

[77] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[78] A. Thomas, A. Gu, T. Dao, A. Rudra, and C. Ré. “Learning compressed transforms with low displacement rank”. In: *Advances in neural information processing systems* 2018 (2018), p. 9052.

[79] A. Titti, S. Squartini, and F. Piazza. “A new time-variant neural based approach for nonstationary and non-linear system identification”. In: *2005 IEEE International Symposium on Circuits and Systems*. IEEE. 2005, pp. 5134–5137.

[80] J. Van Lint, S. Hoogendoorn, and H. J. van Zuylen. “Accurate freeway travel time prediction with state-space neural networks under missing data”. In: *Transportation Research Part C: Emerging Technologies* 13(5-6) (2005), pp. 347–369.

[81] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi. “A bibliography on semiseparable matrices”. In: *Calcolo* 42(3) (2005), pp. 249–270.

[82] P. Viola and M. Jones. “Robust real-time face detection”. In: *International journal of computer vision* 57(2) (2004), pp. 137–154.

[83] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. “Learning structured sparsity in deep neural networks”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, pp. 2082–2090.

[84] B. Wu, D. Wang, G. Zhao, L. Deng, and G. Li. “Hybrid tensor decomposition in neural network compression”. In: *Neural Networks* 132 (2020), pp. 309–320.

[85] Z. Yang, M. Moczulski, M. Denil, N. De Freitas, A. Smola, L. Song, and Z. Wang. “Deep fried convnets”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1476–1483.

[86] J. Zamarreño and P. Vega. “State space neural network. Properties and application”. In: *Neural networks* 11(6) (1998), pp. 1099–1112.

[87] L. Zhao, S. Liao, Y. Wang, Z. Li, J. Tang, and B. Yuan. “Theoretical properties for neural networks with weight matrices of low displacement rank”. In: *international conference on machine learning*. PMLR. 2017, pp. 4082–4090.

[88] X. Zhao, X. Hu, W. Cai, and G. E. Karniadakis. “Adaptive finite element method for fractional differential equations using hierarchical matrices”. In: *Computer Methods in Applied Mechanics and Engineering* 325 (2017), pp. 56–76.

[89] J. G. C. Zuluaga, J. P. Leidig, C. Trefftz, and G. Wolffe. “Deep Reinforcement Learning for Autonomous Search and Rescue”. In: *NAECON 2018 - IEEE National Aerospace and Electronics Conference*. 2018, pp. 521–524.

A. Reprinting Licenses

A.1. Structured Matrices and their Application in Neural Networks: A Survey

The paper [50] is published under the Creative Commons Attribution 4.0[1] International license, which allows reprinting the content. No changes were made to the original publication. The license is reproduced in the following.

[1] https://creativecommons.org/licenses/by/4.0/

Section 1 -- Definitions.

a. Adapted Material means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. Adapter's License means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. Licensor means the individual(s) or entity(ies) granting rights under this Public License.

i. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. You means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

Section 2 -- Scope.

a. License grant.

   1. Subject to the terms and conditions of this Public License,

the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

   a. reproduce and Share the Licensed Material, in whole or in part; and

   b. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

   a. Offer from the Licensor -- Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

   b. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly

reserves any right to collect such royalties.

Section 3 -- License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

 1. If You Share the Licensed Material (including in modified form), You must:

  a. retain the following if it is supplied by the Licensor with the Licensed Material:

   i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

   ii. a copyright notice;

   iii. a notice that refers to this Public License;

   iv. a notice that refers to the disclaimer of warranties;

   v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

  b. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

  c. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

 2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

 3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

 4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 -- Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 -- Disclaimer of Warranties and Limitation of Liability.

a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.

b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION, NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES, COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 -- Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

   1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

   2. upon express reinstatement by the Licensor.

   For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 -- Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 -- Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

===

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the “Licensor.” The text of the Creative Commons public licenses is dedicated to the public domain under the CC0 Public Domain Dedication. Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at creativecommons.org/policies, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at creativecommons.org.

A.2. Backpropagation Through States: Training Neural Networks with Sequentially Semiseparable Weight Matrices

The permission to reprint the paper [51] has been granted through the Copyright Clearance Center's RightsLink® service. The granted license is shown on the following pages.

SPRINGER NATURE LICENSE
TERMS AND CONDITIONS

Oct 20, 2023

This Agreement between Mr. Matthias Kissel ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number	5647110265294
License date	Oct 13, 2023
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	Backpropagation Through States: Training Neural Networks with Sequentially Semiseparable Weight Matrices
Licensed Content Author	Matthias Kissel, Martin Gottwald, Biljana Gjeroska et al
Licensed Content Date	Jan 1, 2022
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	print and electronic
Portion	full article/chapter
Will you be translating?	no
Circulation/distribution	30 - 99

Author of this Springer Nature content	yes
Title	Neural Networks with Sequentially Semiseparable Weight Matrices
Institution name	Technical University of Munich
Expected presentation date	Dec 2023
Requestor Location	Mr. Matthias Kissel [redacted] [redacted] Germany Attn: Mr. Matthias Kissel
Billing Type	Invoice
Billing Address	Mr. Matthias Kissel [redacted] [redacted]
Total	0.00 EUR
Terms and Conditions	

agree that the rights granted to you under this License do not include the right to modify, edit, translate, include in collective works, or create derivative works of the Licensed Material in whole or in part unless expressly stated in your RightsLink Licence Details. You may use the Licensed Material only as permitted under this Agreement and will not reproduce, distribute, display, perform, or otherwise use or exploit any Licensed Material in any way, in whole or in part, except as expressly permitted by this License.

1. 2. You may only use the Licensed Content in the manner and to the extent permitted by these Terms and Conditions, by your RightsLink Licence Details and by any applicable laws.

1. 3. A separate license may be required for any additional use of the Licensed Material, e.g. where a license has been purchased for print use only, separate permission must be obtained for electronic re-use. Similarly, a License is only valid in the language selected and does not apply for editions in other languages unless additional translation rights have been granted separately in the License.

1. 4. Any content within the Licensed Material that is owned by third parties is expressly excluded from the License.

1. 5. Rights for additional reuses such as custom editions, computer/mobile applications, film or TV reuses and/or any other derivative rights requests require additional permission and may be subject to an additional fee. Please apply to journalpermissions@springernature.com or bookpermissions@springernature.com for these rights.

2. Reservation of Rights

Licensor reserves all rights not expressly granted to you under this License. You acknowledge and agree that nothing in this License limits or restricts Licensor's rights in or use of the Licensed Material in any way. Neither this License, nor any act, omission, or statement by Licensor or you, conveys any ownership right to you in any Licensed Material, or to any element or portion thereof. As between Licensor and you, Licensor owns and retains all right, title, and interest in and to the Licensed Material subject to the license granted in Section 1.1. Your permission to use the Licensed Material is expressly conditioned on you not impairing Licensor's or the applicable copyright owner's rights in the Licensed Material in any way.

3. Restrictions on use

3. 1. Minor editing privileges are allowed for adaptations for stylistic purposes or formatting purposes provided such alterations do not alter the original meaning or intention of the Licensed Material and the new figure(s) are still accurate and representative of the Licensed Material. Any other changes including but not limited to, cropping, adapting, and/or omitting material that affect the meaning, intention or moral rights of the author(s) are strictly prohibited.

3. 2. You must not use any Licensed Material as part of any design or trademark.

3. 3. Licensed Material may be used in Open Access Publications (OAP), but any such reuse must include a clear acknowledgment of this permission visible at the same time as the figures/tables/illustration or abstract and which must indicate that the Licensed Material is not part of the governing OA license but has been reproduced with permission. This may be indicated according to any standard referencing system but must include at a minimum 'Book/Journal title, Author, Journal Name (if applicable), Volume (if applicable), Publisher, Year, reproduced

with permission from SNCSC'.

4. STM Permission Guidelines

4. 1. An alternative scope of license may apply to signatories of the STM Permissions Guidelines ("STM PG") as amended from time to time and made available at https://www.stm-assoc.org/intellectual-property/permissions/permissions-guidelines/.

4. 2. For content reuse requests that qualify for permission under the STM PG, and which may be updated from time to time, the STM PG supersede the terms and conditions contained in this License.

4. 3. If a License has been granted under the STM PG, but the STM PG no longer apply at the time of publication, further permission must be sought from the Rightsholder. Contact journalpermissions@springernature.com or bookpermissions@springernature.com for these rights.

5. Duration of License

5. 1. Unless otherwise indicated on your License, a License is valid from the date of purchase ("License Date") until the end of the relevant period in the below table:

Reuse in a medical communications project	Reuse up to distribution or time period indicated in License
Reuse in a dissertation/thesis	Lifetime of thesis
Reuse in a journal/magazine	Lifetime of journal/magazine
Reuse in a book/textbook	Lifetime of edition
Reuse on a website	1 year unless otherwise specified in the License
Reuse in a presentation/slide kit/poster	Lifetime of presentation/slide kit/poster. Note: publication whether electronic or in print of presentation/slide kit/poster may require further permission.
Reuse in conference proceedings	Lifetime of conference proceedings
Reuse in an annual report	Lifetime of annual report
Reuse in training/CME materials	Reuse up to distribution or time period indicated in License
Reuse in newsmedia	Lifetime of newsmedia
Reuse in coursepack/classroom materials	Reuse up to distribution and/or time period indicated in license

6. Acknowledgement

6. 1. The Licensor's permission must be acknowledged next to the Licensed Material in print. In electronic form, this acknowledgement must be visible at the same time as the figures/tables/illustrations or abstract and must be hyperlinked to the journal/book's homepage.

6. 2. Acknowledgement may be provided according to any standard referencing system and at a minimum should include "Author, Article/Book Title, Journal name/Book imprint, volume, page number, year, Springer Nature".

7. Reuse in a dissertation or thesis

7. 1. Where 'reuse in a dissertation/thesis' has been selected, the following terms apply: Print rights of the Version of Record are provided for; electronic rights for use only on institutional repository as defined by the Sherpa guideline (www.sherpa.ac.uk/romeo/) and only up to what is required by the awarding institution.

7. 2. For theses published under an ISBN or ISSN, separate permission is required. Please contact journalpermissions@springernature.com or bookpermissions@springernature.com for these rights.

7. 3. Authors must properly cite the published manuscript in their thesis according to current citation standards and include the following acknowledgement: '*Reproduced with permission from Springer Nature'*.

8. License Fee

You must pay the fee set forth in the License Agreement (the "License Fees"). All amounts payable by you under this License are exclusive of any sales, use, withholding, value added or similar taxes, government fees or levies or other assessments. Collection and/or remittance of such taxes to the relevant tax authority shall be the responsibility of the party who has the legal obligation to do so.

9. Warranty

9. 1. The Licensor warrants that it has, to the best of its knowledge, the rights to license reuse of the Licensed Material. **You are solely responsible for ensuring that the material you wish to license is original to the Licensor and does not carry the copyright of another entity or third party (as credited in the published version).** If the credit line on any part of the Licensed Material indicates that it was reprinted or adapted with permission from another source, then you should seek additional permission from that source to reuse the material.

9. 2. EXCEPT FOR THE EXPRESS WARRANTY STATED HEREIN AND TO THE EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR PROVIDES THE LICENSED MATERIAL "AS IS" AND MAKES NO OTHER REPRESENTATION OR WARRANTY. LICENSOR EXPRESSLY DISCLAIMS ANY LIABILITY FOR ANY CLAIM ARISING FROM OR OUT OF THE CONTENT, INCLUDING BUT NOT LIMITED TO ANY ERRORS, INACCURACIES, OMISSIONS, OR DEFECTS CONTAINED THEREIN, AND ANY IMPLIED OR EXPRESS WARRANTY AS TO MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL LICENSOR BE LIABLE TO YOU OR ANY OTHER PARTY OR ANY OTHER PERSON OR FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL, INDIRECT, PUNITIVE, OR EXEMPLARY DAMAGES, HOWEVER CAUSED, ARISING OUT OF OR IN CONNECTION WITH THE DOWNLOADING, VIEWING OR USE OF THE LICENSED MATERIAL REGARDLESS OF THE FORM OF ACTION, WHETHER FOR BREACH OF CONTRACT, BREACH OF WARRANTY, TORT, NEGLIGENCE, INFRINGEMENT OR OTHERWISE (INCLUDING, WITHOUT LIMITATION, DAMAGES BASED ON LOSS OF PROFITS, DATA, FILES, USE, BUSINESS OPPORTUNITY OR CLAIMS OF THIRD PARTIES), AND WHETHER OR NOT THE PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION APPLIES NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN.

10. Termination and Cancellation

10. 1. The License and all rights granted hereunder will continue until the end of the applicable period shown in Clause 5.1 above. Thereafter, this license will be terminated and all rights granted hereunder will cease.

10. 2. Licensor reserves the right to terminate the License in the event that payment is not received in full or if you breach the terms of this License.

11. General

11. 1. The License and the rights and obligations of the parties hereto shall be construed, interpreted and determined in accordance with the laws of the Federal Republic of Germany without reference to the stipulations of the CISG (United Nations Convention on Contracts for the International Sale of Goods) or to Germany´s choice-of-law principle.

11. 2. The parties acknowledge and agree that any controversies and disputes arising out of this License shall be decided exclusively by the courts of or having jurisdiction for Heidelberg, Germany, as far as legally permissible.

11. 3. This License is solely for Licensor's and Licensee's benefit. It is not for the benefit of any other person or entity.

Questions? For questions on Copyright Clearance Center accounts or website issues please contact springernaturesupport@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777. For questions on Springer Nature licensing please visit https://www.springernature.com/gp/partners/rights-permissions-third-party-distribution

Other Conditions:

Version 1.4 - Dec 2022

Questions? customercare@copyright.com.

A.3. Exploiting Structures in Weight Matrices for Efficient Real-Time Drone Control with Neural Networks

The permission to reprint the paper [52] has been granted through the Copyright Clearance Center's RightsLink® service. The granted license is shown on the following pages.

SPRINGER NATURE LICENSE
TERMS AND CONDITIONS

Oct 20, 2023

This Agreement between Mr. Matthias Kissel ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number	5647110181485
License date	Oct 13, 2023
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	Exploiting Structures in Weight Matrices for Efficient Real-Time Drone Control with Neural Networks
Licensed Content Author	Matthias Kissel, Sven Gronauer, Mathias Korte et al
Licensed Content Date	Jan 1, 2022
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	print and electronic
Portion	full article/chapter
Will you be translating?	no
Circulation/distribution	30 - 99
Author of this Springer Nature content	yes

Title	Neural Networks with Sequentially Semiseparable Weight Matrices
Institution name	Technical University of Munich
Expected presentation date	Dec 2023
Requestor Location	Mr. Matthias Kissel [redacted] [redacted] Germany Attn: Mr. Matthias Kissel
Billing Type	Invoice
Billing Address	Mr. Matthias Kissel [redacted] [redacted]
Total	0.00 EUR
Terms and Conditions	

Springer Nature Customer Service Centre GmbH Terms and Conditions

The following terms and conditions ("Terms and Conditions") together with the terms specified in your [RightsLink] constitute the License ("License") between you as Licensee and Springer Nature Customer Service Centre GmbH as Licensor. By clicking 'accept' and completing the transaction for your use of the material ("Licensed Material"), you confirm your acceptance of and obligation to be bound by these Terms and Conditions.

1. Grant and Scope of License

1. 1. The Licensor grants you a personal, non-exclusive, non-transferable, non-sublicensable, revocable, world-wide License to reproduce, distribute, communicate to the public, make available, broadcast, electronically transmit or create derivative works using the Licensed Material for the purpose(s) specified in your RightsLink Licence Details only. Licenses are granted for the specific use requested in the order and for no other use, subject to these Terms and Conditions. You acknowledge and agree that the rights granted to you under this License do not include the right to modify, edit, translate, include in collective works, or create derivative works of the Licensed Material in whole or in part unless expressly stated in your RightsLink

Licence Details. You may use the Licensed Material only as permitted under this Agreement and will not reproduce, distribute, display, perform, or otherwise use or exploit any Licensed Material in any way, in whole or in part, except as expressly permitted by this License.

1. 2. You may only use the Licensed Content in the manner and to the extent permitted by these Terms and Conditions, by your RightsLink Licence Details and by any applicable laws.

1. 3. A separate license may be required for any additional use of the Licensed Material, e.g. where a license has been purchased for print use only, separate permission must be obtained for electronic re-use. Similarly, a License is only valid in the language selected and does not apply for editions in other languages unless additional translation rights have been granted separately in the License.

1. 4. Any content within the Licensed Material that is owned by third parties is expressly excluded from the License.

1. 5. Rights for additional reuses such as custom editions, computer/mobile applications, film or TV reuses and/or any other derivative rights requests require additional permission and may be subject to an additional fee. Please apply to journalpermissions@springernature.com or bookpermissions@springernature.com for these rights.

2. Reservation of Rights

Licensor reserves all rights not expressly granted to you under this License. You acknowledge and agree that nothing in this License limits or restricts Licensor's rights in or use of the Licensed Material in any way. Neither this License, nor any act, omission, or statement by Licensor or you, conveys any ownership right to you in any Licensed Material, or to any element or portion thereof. As between Licensor and you, Licensor owns and retains all right, title, and interest in and to the Licensed Material subject to the license granted in Section 1.1. Your permission to use the Licensed Material is expressly conditioned on you not impairing Licensor's or the applicable copyright owner's rights in the Licensed Material in any way.

3. Restrictions on use

3. 1. Minor editing privileges are allowed for adaptations for stylistic purposes or formatting purposes provided such alterations do not alter the original meaning or intention of the Licensed Material and the new figure(s) are still accurate and representative of the Licensed Material. Any other changes including but not limited to, cropping, adapting, and/or omitting material that affect the meaning, intention or moral rights of the author(s) are strictly prohibited.

3. 2. You must not use any Licensed Material as part of any design or trademark.

3. 3. Licensed Material may be used in Open Access Publications (OAP), but any such reuse must include a clear acknowledgment of this permission visible at the same time as the figures/tables/illustration or abstract and which must indicate that the Licensed Material is not part of the governing OA license but has been reproduced with permission. This may be indicated according to any standard referencing system but must include at a minimum 'Book/Journal title, Author, Journal Name (if applicable), Volume (if applicable), Publisher, Year, reproduced with permission from SNCSC'.

4. STM Permission Guidelines

4. 1. An alternative scope of license may apply to signatories of the STM Permissions Guidelines ("STM PG") as amended from time to time and made available at https://www.stm-assoc.org/intellectual-property/permissions/permissions-guidelines/.

4. 2. For content reuse requests that qualify for permission under the STM PG, and which may be updated from time to time, the STM PG supersede the terms and conditions contained in this License.

4. 3. If a License has been granted under the STM PG, but the STM PG no longer apply at the time of publication, further permission must be sought from the Rightsholder. Contact journalpermissions@springernature.com or bookpermissions@springernature.com for these rights.

5. Duration of License

5. 1. Unless otherwise indicated on your License, a License is valid from the date of purchase ("License Date") until the end of the relevant period in the below table:

Reuse in a medical communications project	Reuse up to distribution or time period indicated in License
Reuse in a dissertation/thesis	Lifetime of thesis
Reuse in a journal/magazine	Lifetime of journal/magazine
Reuse in a book/textbook	Lifetime of edition
Reuse on a website	1 year unless otherwise specified in the License
Reuse in a presentation/slide kit/poster	Lifetime of presentation/slide kit/poster. Note: publication whether electronic or in print of presentation/slide kit/poster may require further permission.
Reuse in conference proceedings	Lifetime of conference proceedings
Reuse in an annual report	Lifetime of annual report
Reuse in training/CME materials	Reuse up to distribution or time period indicated in License
Reuse in newsmedia	Lifetime of newsmedia
Reuse in coursepack/classroom materials	Reuse up to distribution and/or time period indicated in license

6. Acknowledgement

6. 1. The Licensor's permission must be acknowledged next to the Licensed Material in print. In electronic form, this acknowledgement must be visible at the same time as the figures/tables/illustrations or abstract and must be hyperlinked to the journal/book's homepage.

6. 2. Acknowledgement may be provided according to any standard referencing system and at a minimum should include "Author, Article/Book Title, Journal name/Book imprint, volume, page number, year, Springer Nature".

7. Reuse in a dissertation or thesis

7. 1. Where 'reuse in a dissertation/thesis' has been selected, the following terms apply: Print rights of the Version of Record are provided for; electronic rights for

use only on institutional repository as defined by the Sherpa guideline (www.sherpa.ac.uk/romeo/) and only up to what is required by the awarding institution.

7. 2. For theses published under an ISBN or ISSN, separate permission is required. Please contact journalpermissions@springernature.com or bookpermissions@springernature.com for these rights.

7. 3. Authors must properly cite the published manuscript in their thesis according to current citation standards and include the following acknowledgement: '*Reproduced with permission from Springer Nature'*.

8. License Fee

You must pay the fee set forth in the License Agreement (the "License Fees"). All amounts payable by you under this License are exclusive of any sales, use, withholding, value added or similar taxes, government fees or levies or other assessments. Collection and/or remittance of such taxes to the relevant tax authority shall be the responsibility of the party who has the legal obligation to do so.

9. Warranty

9. 1. The Licensor warrants that it has, to the best of its knowledge, the rights to license reuse of the Licensed Material. **You are solely responsible for ensuring that the material you wish to license is original to the Licensor and does not carry the copyright of another entity or third party (as credited in the published version).** If the credit line on any part of the Licensed Material indicates that it was reprinted or adapted with permission from another source, then you should seek additional permission from that source to reuse the material.

9. 2. EXCEPT FOR THE EXPRESS WARRANTY STATED HEREIN AND TO THE EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR PROVIDES THE LICENSED MATERIAL "AS IS" AND MAKES NO OTHER REPRESENTATION OR WARRANTY. LICENSOR EXPRESSLY DISCLAIMS ANY LIABILITY FOR ANY CLAIM ARISING FROM OR OUT OF THE CONTENT, INCLUDING BUT NOT LIMITED TO ANY ERRORS, INACCURACIES, OMISSIONS, OR DEFECTS CONTAINED THEREIN, AND ANY IMPLIED OR EXPRESS WARRANTY AS TO MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL LICENSOR BE LIABLE TO YOU OR ANY OTHER PARTY OR ANY OTHER PERSON OR FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL, INDIRECT, PUNITIVE, OR EXEMPLARY DAMAGES, HOWEVER CAUSED, ARISING OUT OF OR IN CONNECTION WITH THE DOWNLOADING, VIEWING OR USE OF THE LICENSED MATERIAL REGARDLESS OF THE FORM OF ACTION, WHETHER FOR BREACH OF CONTRACT, BREACH OF WARRANTY, TORT, NEGLIGENCE, INFRINGEMENT OR OTHERWISE (INCLUDING, WITHOUT LIMITATION, DAMAGES BASED ON LOSS OF PROFITS, DATA, FILES, USE, BUSINESS OPPORTUNITY OR CLAIMS OF THIRD PARTIES), AND WHETHER OR NOT THE PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION APPLIES NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN.

10. Termination and Cancellation

10. 1. The License and all rights granted hereunder will continue until the end of the applicable period shown in Clause 5.1 above. Thereafter, this license will be terminated and all rights granted hereunder will cease.

10. 2. Licensor reserves the right to terminate the License in the event that payment is not received in full or if you breach the terms of this License.

11. General

11. 1. The License and the rights and obligations of the parties hereto shall be construed, interpreted and determined in accordance with the laws of the Federal Republic of Germany without reference to the stipulations of the CISG (United Nations Convention on Contracts for the International Sale of Goods) or to Germany´s choice-of-law principle.

11. 2. The parties acknowledge and agree that any controversies and disputes arising out of this License shall be decided exclusively by the courts of or having jurisdiction for Heidelberg, Germany, as far as legally permissible.

11. 3. This License is solely for Licensor's and Licensee's benefit. It is not for the benefit of any other person or entity.

Questions? For questions on Copyright Clearance Center accounts or website issues please contact springernaturesupport@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777. For questions on Springer Nature licensing please visit https://www.springernature.com/gp/partners/rights-permissions-third-party-distribution

Other Conditions:

Version 1.4 - Dec 2022

Questions? customercare@copyright.com.

A.4. Deep Convolutional Neural Networks with Sequentially Semiseparable Weight Matrices

The permission to reprint the paper [48] is permitted by the copyright transfer agreement, as shown in the following permission letter and agreement details.

RE: Copyright Form ESANN

ESANN <esann@uclouvain.be>

Do, 06.06.2024 15:35

An:Matthias Kissel <matthias.kissel@outlook.de>

Dear Mr Kissel,

The copyright provisions for the ESANN conference may be found at https://www.esann.org/author_guidelines (bottom of the page). You have of course the right to reuse your own paper, for your PhD thesis of any other distribution.

Best regards,
Michel

ESANN 2024

(https://www.esann.org/)

Guidelines for authors

Instructions for authors

- Papers **must not exceed 6 pages**, including figures and references.
- Papers submitted to one of the special sessions (see topics and special sessions) must be identified on the paper submission form.
- The review process is single blind.
- LaTeX and Word style files are available. They must be used for generating the PDF file (see below). However the use of LaTeX is strongly encouraged, and the use of Word is discouraged as the latter usually generates many difficulties when the final proceedings are generated.
- **Printing area must be 12.2 x 19.3 cm, centered on the A4 page. Left, Right, top and bottom margins will thus be respectively 4.4, 4.4, 5.2 and 5.2 cm.** Complying with these margins and centering the text on the A4 sheets is mandatory: the manuscript will be reproduced in its original size in the proceedings, and margins will be cut to the book format.
- The pages of the manuscript will not be numbered (numbering decided by the editor).

LaTeX style file

Authors may download a LaTeX template and use its to typeset their manuscript.

Warning: although these templates have been succesfully used for years, they have some limitations. We regret that we cannot give support for their use. Furthermore, it is the responsibility of the authors to check if a submission produced with the help of these files does correspond to the instructions for authors, in particular what concerns the margin requirements. If necessary, margins should be adapted to produce the required format.

The .tex file itself contains important instructions. **In particular, do not forget to set the page size in your software (LaTeX, dvi2ps, and any other file generator used to produce the PDF file) to A4, not to Letter**.

Please click on esannV2.tex (/system/files/downloads/esannV2.tex) to get the .tex source file to be used to typeset your manuscript, and on esannV2.cls (/system/files/downloads/esannV2.cls) to get the style file itself. An example of PDF file is available: click on esannV2.pdf (/system/files/downloads/esannV2.pdf). If you wish to recompile the esannV2.tex file, you should have the ESANN2005BW.eps (/system/files/downloads/ESANN2005BW.eps) figure too.

Word style file (not encouraged!)

Authors may download a Word template to typeset their manuscript; this template is available in .doc and .dot formats: please click on esannV2.doc (/system/files/downloads/esannV2.doc) or on esannV2.dot (/system/files/downloads/esannV2.dot).

Warning: although these templates have been succesfully used for years, they have some limitations. We regret that we cannot give support for their use. Furthermore, it is the responsibility of the authors to check if a submission produced with the help of these files does correspond to the instructions for authors, in particular what concerns the margin requirements. If necessary, margins should be adapted to produce the required format.

The style file itself contains important instructions. **In particular, do not forget to set the page size in your software (Word, and also any file generator to produce the PDF file) to A4, not to Letter.**

Authorized file formats

Your manuscript should be uploaded in PDF format (or, if needed, in PS format); any other format, including Word, is not accepted. Please use up-to-date professional software to generate your PS or PDF file.

Before uploading your file check that the format is A4 (and not Letter). Indeed some PDF generators generate by default a file in Letter format. If this is the case, please change the format to A4 both in your Word or LateX software *and* in the PDF generator, before uploading your file.

PDF settings must be set to 600 dpi..

Ethics, copyright transfer and registration

By submitting your paper (both the submission and the final version),

- you confirm that, in case of acceptance of your submission, at least one of the authors will register and pay the registration fee before 19.08.2024 (one full registration is needed for each accepted paper - see details on registration page for authors presenting multiple papers);
- you confirm that one of the registered authors will attend the conference and present the paper; the ESANN conference applies a strict policy about the presentation of accepted papers during the conference: authors of accepted papers who do not show up at the conference will be blacklisted for future ESANN and other conferences in the field;
- you confirm that you comply with ESANN publication ethics (/ethics);
- you confirm that the same work has not been published or submitted elsewhere (conference or journal);
- you confirm that the same work will not be submitted to any other conference or journal before the notification of decision of your submission to ESANN;
- you accept not to distribute your paper before the conference (in particular, you will not make it available on a web page or in a publication repository);

- you give the rights to distribute and reproduce your paper in any form (paper and electronic) to the ESANN organization (d-side s.a.). In particular, you accept that your paper will be printed in the proceedings, made available on the web and on any possible electronic support. You keep the rights on your paper and you may freely distribute it and archive it in any repository, provided that full reference is given. Once it will be made available on the ESANN proceedings web pages, only this version may be distributed (it will contain the full references inserted on each page of the paper).

A.5. Neural Networks comprising Sequentially Semiseparable Matrices with one dimensional State Variable are Universal Approximators

The permission to reprint is given by the license agreement, as shown in the following.

Licence to Publish Proceedings Papers

SPRINGER NATURE

Licensee	Springer Nature Switzerland AG	(the 'Licensee')
Title of the Proceedings Volume/Edited Book or Conference Name:	N/A*	(the 'Volume')
Volume Editor(s) Name(s):	N/A*	
Proposed Title of the Contribution:	Neural Networks comprising Sequentially Semiseparable Matrices with one dimensional State Variable are Universal Approximators*	(the 'Contribution')
Series: The Contribution may be published in the following series	A Springer Nature Computer Science book series (CCIS, LNAI, LNBI, LNBIP or LNCS)	
Author(s) Full Name(s):	Matthias Kissel and Klaus Diepold*	(the 'Author')

When Author is more than one person the expression "Author" as used in this Agreement will apply collectively unless otherwise indicated.

Corresponding Author Name:	Matthias Kissel*	
Instructions for Authors	https://resource-cms.springernature.com/springer-cms/rest/v1/content/19242230/data/	(the 'Instructions for Authors')

*Information added on 01.07.2024 for completeness.

1 Grant of Rights

a) For good and valuable consideration, the Author hereby grants to the Licensee the perpetual, exclusive, world-wide, assignable, sublicensable and unlimited right to: publish, reproduce, copy, distribute, communicate, display publicly, sell, rent and/or otherwise make available the contribution identified above, including any supplementary information and graphic elements therein (e.g. illustrations, charts, moving images) (the 'Contribution') in any language, in any versions or editions in any and all forms and/or media of expression (including without limitation in connection with any and all end-user devices), whether now known or developed in the future. Without limitation, the above grant includes: (i) the right to edit, alter, adapt, adjust and prepare derivative works; (ii) all advertising and marketing rights including without limitation in relation to social media; (iii) rights for any training, educational and/or instructional purposes; (iv) the right to add and/or remove links or combinations with other media/works; and (v) the right to create, use and/or license and/or sublicense content data or metadata of any kind in relation to the Contribution (including abstracts and summaries) without restriction. The above rights are granted in relation to the Contribution as a whole or any part and with or in relation to any other works.

b) Without limiting the rights granted above, Licensee is granted the rights to use the Contribution for the purposes of analysis, testing, and development of publishing- and research-related workflows, systems, products, projects, and services; to confidentially share the Contribution with select third parties to do the same; and to retain and store the Contribution and any associated correspondence/files/forms to maintain the historical record, and to facilitate research integrity investigations. The grant of rights set forth in this clause (b) is irrevocable.

c) If the Licensee elects not to publish the Contribution for any reason, all publishing rights under this Agreement as set forth in clause 1a above will revert to the Author.

2 Copyright

Ownership of copyright in the Contribution will be vested in the name of the Author. When reproducing the Contribution or extracts from it, the Author will acknowledge and reference first publication in the Volume.

3 Use of Contribution Versions

a) For purposes of this Agreement: (i) references to the "Contribution" include all versions of the Contribution; (ii) "Submitted Manuscript" means the version of the Contribution as first submitted by the Author prior to peer review; (iii) "Accepted Manuscript" means the version of the Contribution accepted for publication, but prior to copy-editing and typesetting; and (iv) "Version of Record" means the version of the Contribution published by the Licensee, after copy-editing and typesetting. Rights to all versions of the Manuscript are granted on an exclusive basis, except for the Submitted Manuscript, to which rights are granted on a non-exclusive basis.

b) The Author may make the Submitted Manuscript available at any time and under any terms (including, but not limited to, under a CC BY licence), at the Author's discretion. Once the Contribution has been published, the Author will include an acknowledgement and provide a link to the Version of Record on the publisher's website: "This preprint has not undergone peer review (when applicable) or any post-submission improvements or corrections. The Version of Record of this contribution is published in [insert volume title], and is available online at https://doi.org/[insert DOI]".

c) The Licensee grants to the Author (i) the right to make the Accepted Manuscript available on their own personal, self-maintained website immediately on acceptance, (ii) the right to make the Accepted Manuscript available for public release on any of the following twelve (12) months after first publication (the "Embargo Period"): their employer's internal website; their institutional and/or funder repositories. Accepted Manuscripts may be deposited in such repositories immediately upon acceptance, provided they are not made publicly available until after the Embargo Period.
The rights granted to the Author with respect to the Accepted Manuscript are subject to the conditions that (i) the Accepted Manuscript is not enhanced or substantially reformatted by the Author or any third party, and (ii) the Author includes on the Accepted Manuscript an acknowledgement in the following form, together with a link to the published version on the publisher's website: "This version of the contribution has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/[insert DOI]. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms". Under no circumstances may an Accepted Manuscript be shared or distributed under a Creative Commons or other form of open access licence.
Any use of the Accepted Manuscript not expressly permitted under this subclause (c) is subject to the Licensee's prior consent.

d) The Licensee grants to Author the following non-exclusive rights to the Version of Record, provided that, when reproducing the Version of Record or extracts from it, the Author acknowledges and references first publication in the Volume according to current citation standards. As a minimum, the acknowledgement must state: "First published in [Volume, page number, year] by Springer Nature".

i. to reuse graphic elements created by the Author and contained in the Contribution, in presentations and other works created by them;

ii. the Author and any academic institution where they work at the time may reproduce the Contribution for the purpose of course teaching (but not for inclusion in course pack material for onward sale by libraries and institutions);

iii. to reuse the Version of Record or any part in a thesis written by the same Author, and to make a copy of that thesis available in a repository of the Author(s)' awarding academic institution, or other repository required by the awarding academic institution. An acknowledgement should be included in the citation: "Reproduced with permission from Springer Nature";

iv. to reproduce, or to allow a third party to reproduce the Contribution, in whole or in part, in any other type of work (other than thesis) written by the Author for distribution by a publisher after an embargo period of 12 months; and

v. to publish an expanded version of their Contribution provided the expanded version (i) includes at least 30% new material (ii) includes an express statement specifying the incremental change in the expanded version (e.g., new results, better description of materials, etc.).

4 Warranties & Representations

Author warrants and represents that:

a)

i. the Author is the sole copyright owner or has been authorised by any additional copyright owner(s) to grant the rights defined in clause 1,

ii. the Contribution does not infringe any intellectual property rights (including without limitation copyright, database rights or trade mark rights) or other third party rights and no licence from or payments to a third party are required to publish the Contribution,

iii. the Contribution has not been previously published or licensed, nor has the Author committed to licensing any version of the Contribution under a licence inconsistent with the terms of this Agreement,

iv. if the Contribution contains materials from other sources (e.g. illustrations, tables, text quotations), Author has obtained written permissions to the extent necessary from the copyright holder(s), to license to the Licensee the same rights as set out in clause 1 but on a non-exclusive basis and without the right to use any graphic elements on a stand-alone basis and has cited any such materials correctly;

b) all of the facts contained in the Contribution are according to the current body of research true and accurate;

c) nothing in the Contribution is obscene, defamatory, violates any right of privacy or publicity, infringes any other human, personal or other rights of any person or entity or is otherwise unlawful and that informed consent to publish has been obtained for any research participants;

d) nothing in the Contribution infringes any duty of confidentiality owed to any third party or violates any contract, express or implied, of the Author;

e) all institutional, governmental, and/or other approvals which may be required in connection with the research reflected in the Contribution have been obtained and continue in effect;

f) all statements and declarations made by the Author in connection with the Contribution are true and correct;

g) the signatory who has signed this Agreement has full right, power and authority to enter into this Agreement on behalf of all of the Authors; and

h) the Author complies in full with: i. all instructions and policies in the Instructions for Authors, ii. the Licensee's ethics rules (available at https://www.springernature.com/gp/authors/book-authors-code-of-conduct), as may be updated by the Licensee at any time in its sole discretion.

5 Cooperation

a) The Author will cooperate fully with the Licensee in relation to any legal action that might arise from the publication of the Contribution, and the Author will give the Licensee access at reasonable times to any relevant accounts, documents and records within the power or control of the Author. The Author agrees that any Licensee affiliate through which the Licensee exercises any rights or performs any obligations under this Agreement is intended to have the benefit of and will have the right to enforce the terms of this Agreement.

b) Author authorises the Licensee to take such steps as it considers necessary at its own expense in the Author's name(s) and on their behalf if the Licensee believes that a third party is infringing or is likely to infringe copyright in the Contribution including but not limited to initiating legal proceedings.

6 Author List

Changes of authorship, including, but not limited to, changes in the corresponding author or the sequence of authors, are not permitted after acceptance of a manuscript.

7 Post Publication Actions

The Author agrees that the Licensee may remove or retract the Contribution or publish a correction or other notice in relation to the Contribution if the Licensee determines that such actions are appropriate from an editorial, research integrity, or legal perspective.

8 Controlling Terms

The terms of this Agreement will supersede any other terms that the Author or any third party may assert apply to any version of the Contribution.

9 Governing Law

This Agreement shall be governed by, and shall be construed in accordance with, the laws of Switzerland. The courts of Zug, Switzerland shall have the exclusive jurisdiction.

Signed for and on behalf of the Author	Print Name:	Date:
[redacted]	Matthias Kissel	26.7.2023

Address:	Arcisstr. 21, 80333 Munich, (TUM, Chair of Data Processing)
Email:	Matthias.Kissel@tum.de

Springer Nature Switzerland AG, Gewerbestrasse 11, 6330 Cham, Switzerland
ER_Book_ProceedingsPaper_LTP_ST_v.1.0 (10_2021)

B. Acceptance Letter Universal Approximation Theorem Paper

SCEFA@ECML-PKDD23: review results

Microsoft CMT <email@msr-cmt.org>

Mi 12.07.2023 19:07

An:Matthias Kissel <matthias.kissel@tum.de>;

Dear Matthias Kissel,

Paper ID: 9
Title: Neural Networks comprising Sequentially Semiseparable Matrices with one dimensional State Variable are Universal Approximators

We are pleased to inform you that the above-listed manuscript has been accepted for presentation at ECML-PKDD workshop track: Simplification, Compression, Efficiency and Frugality for Artificial Intelligence (SCEFA): congratulations!

The reviewers' comments on your paper are available and served as part of the basis for the Technical Committee's decision.

Please be informed that, in case you have received "Revision" as a score, you are required to submit a new, improved version of the paper, including the reviewers and the meta-reviewer comments, by the 18th July AoE. Failing in addressing the comments or in submitting the improved version of the paper per time will result in a desk rejection.

For all the papers having as score “Accept”, the authors are invited to improve them accounting for the reviews received. Further instructions on the camera-ready submission will follow in the next days. Be aware that the presentation of the accepted papers should be in presence, and by at least one registered author.

Thank you for your research, and your desire to share it with the ECML-PKDD Community. With your support and collaboration, we sincerely believe that the workshop on Simplification, Compression, Efficiency and Frugality for Artificial Intelligence will be an inclusive and exciting experience for all and will help spread your research outcome.

Sincerely yours,
Enzo Tartaglione, Attilio Fiandrotti and Giovanna Varni
SCEFA Track Chairs