Parameter Efficient Neural Networks With Singular Value Decomposed Kernels

David Vander Mijnsbrugge, Femke Ongenae, and Sofie Van Hoecke

***Abstract*— Traditionally, neural networks are viewed from the perspective of connected neuron layers represented as matrix multiplications. We propose to compose these weight matrices from a set of orthogonal basis matrices by approaching them as elements of the real matrices vector space under addition and multiplication. Making use of the Kronecker product for vectors, this composition is unified with the singular value decomposition (SVD) of the weight matrix. The orthogonal components of this SVD are trained with a descent curve on the Stiefel manifold using the Cayley transform. Next, update equations for the singular values and initialization routines are derived. Finally, acceleration for stochastic gradient descent optimization using this formulation is discussed. Our proposed method allows more parameter-efficient representations of weight matrices in neural networks. These decomposed weight matrices achieve maximal performance in both standard and more complicated neural architectures. Furthermore, the more parameter-efficient decomposed layers are shown to be less dependent on optimization and better conditioned. As a tradeoff, training time is increased up to a factor of 2. These observations are consequently attributed to the properties of the method and choice of optimization over the manifold of orthogonal matrices.**

***Index Terms*— Cayley transform, Kronecker product, neural networks, singular value decomposition (SVD), Stiefel manifold, vector space.**

I. Introduction

ARTIFICIAL neural networks are based on the mathematical representation (McCulloch–Pitts model) of artificial neurons [1]. Stacking layers of these neurons separated by nonlinear activation functions $\sigma(\cdot)$ creates a model, called the multilayer perceptron, that allows nonlinear modeling [2]. To obtain optimal weights for the network, these models have to be trained. In supervised learning, an appropriate objective function $\mathcal{L}$ is minimized over a set of training examples to achieve this. By using a matrix multiplication to represent the connections between neurons, the full problem statement for a single layer is written as follows:

$$\min_{\mathbf{W},\vec{b}} \mathcal{L}(\vec{y}(\vec{x}),\ \hat{\vec{y}}) \tag{1}$$

$$\vec{y}(\vec{x}) = \sigma(\mathbf{W}\vec{x} + \vec{b}). \tag{2}$$

Manuscript received 13 August 2020; revised 9 July 2021 and 4 November 2021; accepted 22 November 2021. Date of publication 23 December 2021; date of current version 1 September 2023. This work was supported by the Flemish Government through the Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen Programme. *(Corresponding author: David Vander Mijnsbrugge.)*

The authors are with the IDLab, Ghent University–imec, 9052 Gent, Belgium (e-mail: david.vandermijnsbrugge@ugent.be).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2021.3130756.

Digital Object Identifier 10.1109/TNNLS.2021.3130756

At the core of this process are the weight matrices or kernels $\mathbf{W}$ that transform each layer before activation. This kernel describes a set of linear equations between a hidden state vector and the input of the layer. From a mathematical standpoint, this kernel is contained in the set of all real matrices $\mathbb{R}^{M\times N}$, which is a vector space under addition and scalar multiplication [3]. Consequently, each weight matrix can be described as a linear combination of basis matrices $\{\mathbf{E}_i\}$ with $\text{span}\{\mathbf{E}_i\} = \mathbb{R}^{M\times N}$ as formulated in (3). By taking a basis, known as either the standard or natural basis [see (4], the before mentioned neuron interpretation is recovered

$$\mathbf{W} = \sum_i s_i \mathbf{E}_i \tag{3}$$

$$\mathbf{E}_i = \delta_{i,j}^{N\times M} |\ \ \forall j \leq N \times M. \tag{4}$$

In this natural basis, each basis matrix corresponds to an adjacency matrix for the connection between an input and output neuron where each coefficient s_i represents the weight of this connection. From the traditional viewpoint, these basis matrices are fixed and only the coefficients are trained. We propose a new perspective given the basis decomposition, which allows to look at the weight matrix and considers it as a representation of the relationship between two entities. These entities are described using embeddings (Definition 1) and are commonly known from the natural language processing (NLP) domain [4] where they translate words or language entities into mathematical vectors.

Definition 1: Embeddings are vector representations of either categorical or continuous variables, often lower dimensional than the original one.

Deep learning embeddings have been observed to obey semantic rules imposed by the linguistic rules present in the data. These semantic rules can be expressed as relations between entities as in (5). Due to the linearity of the matrix multiplication, we stipulate that these semantic properties can be a direct consequence of the weight matrix representing the relationship as portrayed by (6) in Example 1.

Example 1: Given that the embeddings of the words "Girl"/"Woman" and "young" obey the given relation

$$\vec{\text{Woman}} + \vec{\text{Young}} \sim \vec{\text{Girl}}. \tag{5}$$

This implies the following rules of relations:

$$\textbf{Is\_Woman} + \textbf{Is\_Young} \sim \textbf{Is\_Girl}. \tag{6}$$

Generalizing this kind of decomposition leads to the same decomposition as in (3), where each E_i represents a specific

relation and s_i its strength. This interpretation strays away from the connected neuron interpretation by making both the coefficients and basis matrices trainable. Now, the transformation represented by the kernel $\mathbf{W}$ can be seen as a combination of basis transformations and their respective importance or strength, rather than neuron connections and their individual weight.

The objective of this article is to introduce this interpretation of the weight matrix and formalize it into a framework fitting with the current state of the art on deep learning. We also show the advantages of this method, i.e., improved parameter efficiency and generalization while maintaining performance, through thorough benchmark evaluations.

The remainder of this article is organized as follows. First, Section II presents the related work. Section III introduces the concept of Kronecker product to make both the basis matrices and coefficients trainable with a feasible amount of parameters. In this section, also the associated training algorithm is derived. Finally, the presented method is evaluated on benchmark datasets and the results are consequently linked to the properties of the method in Section IV. Section V presents the major conclusions and future work.

II. Related Work

The concept of learning sets of basis blocks is a fundamental concept of dictionary learning and is often joined with sparsity constraints [5]. However, to the best of our knowledge, such an approach has not yet been explored for neural network layers. The resulting construction of weight matrices using singular value decomposition (SVD) has been proposed in the context of recurrent neural networks using householder matrices [6]. Even though this work extends to general matrices, there is no mention of acceleration or initialization. Furthermore, it does not make use of the large body of work regarding optimization on the Stiefel manifold with the Cayley transform [7]–[11].

On the other hand, in [12], the Cayley transform and corresponding descent curve are effectively used to model unitary matrices in recurrent neural networks. In this work, a similar approach is taken for orthogonal matrices, but in the context of optimizing the orthogonal components of an SVD. Coincidentally, the SVD has been used to investigate the properties of neural networks, such as bounding the singular values for improved training [13] or using the SVD in order to reduce the latent space to a minimal effective dimension [14]. These works use the SVD retrospectively, while in this work, the decomposition is proactively taken as the starting point.

III. Approach

Starting from the basis decomposition in (3), the proposed method will treat each layer by storing, initializing, and training the basis components and coefficients independently. In the forward pass, the weights are assembled from the decomposition and used to calculate the output. In the backward pass, each component is updated separately according to specific update equations. Our approach is divided into four parts corresponding to the following. First, by using a matrix decomposition for the basis matrices, a specific form for basis decomposition is proposed. Second, corresponding to the update, equations for both the basis matrices and the coefficients are derived. Third, initialization routines for each component are constructed. Finally and fourth, stochastic acceleration for each update equation is discussed.

A. Basis Matrix Decomposition

The interpretation from Section I requires making every weight in (3) trainable, resulting in $\mathcal{O}(R \cdot N \cdot M)$ parameters. Here, R is the size of the set of basis matrices. To make this number of parameters feasible, a significant reduction of these parameters per basis matrix is required. Using the concept of the Kronecker product [15] of two vectors [see (7)], each basis matrix is constructed with only $\mathcal{O}(N+M)$ instead of $\mathcal{O}(N \cdot M)$ parameters

$$\begin{aligned} \mathbf{E}_i &= \vec{u}_i \otimes \vec{v}_i \\ \left[\vec{a} \otimes \vec{b}\right]_{ij} &= \vec{a}_i \cdot \vec{b}_j. \end{aligned} \tag{7}$$

This reduction in parameters allows the whole decomposition to consist of $\mathcal{O}(R \cdot [N + M + 1])$ parameters by substituting (7) into (3). Consequently, if the sets of vectors $\{\vec{u}_i\}$ and $\{\vec{v}_i\}$ represented by $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, the resulting form corresponds to the SVD of $\mathbf{W}$, as shown in the following equation:

$$\mathbf{W} = \sum_i s_{ii}[\vec{u}_i \otimes \vec{v}_i] = \mathbf{U}\mathbf{S}\mathbf{V}^T. \tag{8}$$

We can give a straightforward interpretation to this result. The hyperparameter R represents the dimension of the subspaces in which most of the embeddings that are connected by the relation reside. The orthogonality of $\mathbf{U}$ and $\mathbf{V}$ ensures that also the set of basis matrices is orthonormal [see (9)] under the Frobenius norm, as derived in Appendix A-D. This changes the interpretation from the orthonormal basis sets $\{\vec{u}_i\}$ and $\{\vec{v}_i\}$ of the embeddings into a single orthonormal basis set for the relation, which is the same change of interpretation provided in Example 1. Furthermore, the hyperparameter R also grants insight into the cases when the parameter reduction will be useful, namely, in the case where the input and output data space can be adequately described by an R-dimensional subspace of $\mathbb{R}^N$ and $\mathbb{R}^M$, respectively, or equivalently when $\mathbf{W}$ can be adequately described by an R-dimensional subspace of $\mathbb{R}^{N \times M}$

$$< \mathbf{E}_i, \mathbf{E}_j >= \delta_{ij} | \quad \forall E_i, E_j. \tag{9}$$

Parallel to this interpretation, the approach also addresses a recurring problem in neural networks, i.e., the ill-conditioned weight matrices. Ill-conditioned matrices give rise to large fluctuations of the output of the linear system with respect to small fluctuations of the input. This property translates into saddle points in the loss function, which leads to more confusion during training and less stable networks [16]. The conditioning number κ and the fraction of the largest and smallest singular values of the kernel, given by (10), represent this property. Large values of this conditioning number correspond to ill-conditioned matrices

$$\kappa = \frac{\sigma_{\max}}{\sigma_{\min}} = \frac{\max(\mathbf{S})}{\min(\mathbf{S})}. \tag{10}$$

Regularization of singular values has already been shown to improve training time and performance [13]. Our proposed method inherently contains this regularization as standard weight regularization of the network on the decomposition coefficients. In terms of κ, this means that regularizing the weights in $\mathbf{S}$ leads to soft upper and lower bounds on κ and consequently avoids ill-conditioned matrices.

B. Update Equations

To effectively train the decomposed layer, there needs to be a set of update equations for the weights. These weights are most commonly optimized using stochastic gradient descent (SGD) [17]. At each iteration k, SGD adds the gradient of the objective function with respect to the weights at that iteration ($\nabla_{w^k}\mathcal{L}$), scaled by a learning constant α. The standard form for an SGD weight update step is given in the following equation:

$$w^{k+1} = w^k + \alpha \nabla_{w^k}\mathcal{L}. \tag{11}$$

Next, update equations for $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$ in the form of (11) are derived. In light of the general decomposition in (3), the update equations for both $\mathbf{U}$ and $\mathbf{V}$, corresponding to the basis matrices, and $\mathbf{S}$, regarding the coefficients, are considered separately. This section starts with defining the necessary tools and then constructs these update equations for the orthogonal matrices in the SVD. Using these update equations, consequent ones for the singular values are derived.

1) Orthogonal Matrices: By construction, the matrices $\mathbf{U}$ and $\mathbf{V}$ both describe a set of orthonormal basis vectors in a rowwise fashion. To preserve this property, update equations that generate orthogonal matrices at each iteration are required. Definition 2 describes the manifold on which these matrices lie.

Definition 2: The Stiefel manifold [see (12)] describes the set of all the orthogonal matrices under the canonical metric (13), where $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are elements of the tangent space of $\mathbf{X}$ ($\mathcal{T}_{\mathbf{X}}\mathbf{V}_{N,R}$) [18]

$$\mathbf{V}_{N,R} = \{\mathbf{X} \in \mathbb{R}^{N\times R} : \mathbf{X}^T\mathbf{X} = \mathbf{I}_R\} \tag{12}$$

$$< \mathbf{Z}_1, \mathbf{Z}_2 >_c = \mathrm{Tr}\left(\mathbf{Z}_1^T \cdot \left[\mathbf{I} - \frac{1}{2}\mathbf{X}\mathbf{X}^T\right] \cdot \mathbf{Z}_2\right). \tag{13}$$

Optimization of the Stiefel manifold can be done via the geodesic described by the matrix exponential. However, for computational reasons, it is better to use quasi-geodesic methods [7], [10]–[12]. This quasi-geodesic is a Padé approximation of the matrix exponential, called the Cayley transform, described in Theorem 1

Theorem 1: The Cayley transform [see (14)] of any skew-symmetric matrix is orthogonal[1]

$$\phi_\nu(\mathbf{X}) = \left(\mathbf{I} - \frac{\nu}{2}\mathbf{X}\right)^{-1} \cdot \left(\mathbf{I} + \frac{\nu}{2}\mathbf{X}\right)$$
$$\phi_\nu(\mathbf{X})^T \cdot \phi_\nu(\mathbf{X}) = \mathbf{I} \,|\, \mathbf{X}^T = -\mathbf{X}. \tag{14}$$

Definition 3 describes a parameterized descent curve on the Stiefel manifold for any objective function $\mathcal{L}$ using this transform.

[1]Proof in Appendix A-A.

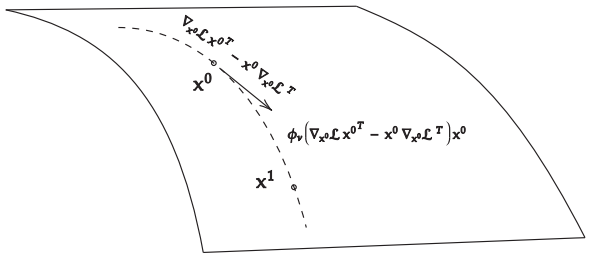


Fig. 1. Representation of descent curve on the Stiefel manifold.

Definition 3: Given a orthogonal matrix $\mathbf{X}$ and an objective function $\mathcal{L}(\mathbf{X})$, $\mathbf{Y}(\nu)$ is a parameterized descent curve for $\mathcal{L}(\mathbf{X})$ on $\mathbf{V}_{N,R}$ [18]

$$\mathbf{Y}(\nu) = \phi_\nu(\mathbf{W}(\mathbf{X}, \nabla_{\mathbf{X}}\mathcal{L}))\mathbf{X} \tag{15}$$

$$\mathbf{W}(\mathbf{X}, \nabla_{\mathbf{X}}\mathcal{L}) = \nabla_{\mathbf{X}}\mathcal{L}\mathbf{X}^T - \mathbf{X}\nabla_{\mathbf{X}}\mathcal{L}^T. \tag{16}$$

This descent curve is directly dependent on the derivative of the specified objective function $\nabla_{\mathbf{X}}\mathcal{L}$. It is used to define the action on $\mathbf{X}$ as $\nabla_{\mathbf{X}}\mathcal{L}\mathbf{X}^T - \mathbf{X}\nabla_{\mathbf{X}}\mathcal{L}^T$. This is a vector in the tangent space of the Stiefel manifold. Consequently, the action is transformed by the Cayley transform into an orthogonal matrix since the action is a skew-symmetric matrix. Due to the orthogonality restriction and the metric, this descent curve can be seen as a search over a surface in the direction that minimizes the loss function. This is visually represented in Fig. 1. Optimization using this descent curve normally uses line search [10], [18] to determine step sizes according to the Armijo–Wolfe conditions, as described in Definition 4 [19], [20].

Definition 4: The Armijo–Wolfe conditions for curvilinear search are

$$\mathcal{L}(\mathbf{Y}(\nu)) < \mathcal{L}(\mathbf{Y}(0)) + c_1 \cdot \nu \cdot \nabla_\nu\mathcal{L}(\mathbf{Y}(0)) \tag{17}$$

$$\nabla_\nu\mathcal{L}(\mathbf{Y}(\nu)) > c_2 \cdot \nabla_\nu\mathcal{L}(\mathbf{Y}(0)) \tag{18}$$

$$0 < c_1 < c_2 < 1. \tag{19}$$

These conditions are to be interpreted as an upper and lower bound on the step sizes, respectively, defined by two coefficients c_1 and c_2. As derived in Appendix A-C, they can be evaluated using the chain rule and the directional derivative of the Cayley transform with respect to ν given in the following equation:

$$\nabla_\nu\mathcal{L}(\phi_\nu(\mathbf{X})) = \mathrm{Tr}(\nabla_{\phi_\nu(\mathbf{X})}\mathcal{L} \cdot \nabla_\nu(\phi_\nu(\mathbf{X})))$$
$$\nabla_\nu(\phi_\nu(\mathbf{X})) = -\frac{1}{2}\left(\mathbf{I} + \frac{\nu}{2}\mathbf{X}\right)^{-1} \cdot \mathbf{X} \cdot (\mathbf{X} + \phi_\nu(\mathbf{X})). \tag{20}$$

To obtain the most acceptable step sizes, the coefficients are often taken to include a large portion of the interval $[0, 1]$, e.g., $c_1 = 0.1$ and $c_2 = 0.9$. Doing a line search for each matrix would lead to $2L$ line searches at each training step for a network with L layers. Using such a method at each training iteration would thus increase training time with a factor proportional to the amount of iterations needed for these searches to converge. Vaswani *et al.* [21] noted that these methods converge faster at the cost of computational time. In the following derivation, we omit this line search for the

benefit of training time. Note that using line search can easily be added when striving for optimal convergence irrespective of training time.

Using the descent curve with a constant step ν solves this problem by suboptimally choosing a value for ν. Equivalently, this is optimizing both $\mathbf{U}$ and $\mathbf{V}$ by taking equidistant steps on the optimization surface according to the descent curve. Equations (21) and (22) hold for $\mathbf{U}$ and $\mathbf{V}$, respectively

$$\mathbf{U}^{k+1} = \phi_\nu(\mathbf{U}^k, \nabla_{\mathbf{U}^k}\mathcal{L})\mathbf{U}^k \tag{21}$$
$$\mathbf{V}^{k+1} = \phi_\nu(\mathbf{V}^k, \nabla_{\mathbf{V}^k}\mathcal{L})\mathbf{V}^k. \tag{22}$$

The calculation of (21) and (22) requires the inversion of matrices with dimensions $N \times N$ and $M \times M$, respectively. Theorem 2 manipulates this calculation into a term χ_ν that requires a $2R \times 2R$ matrix inversion.

Theorem 2: The Cayley transform ϕ_ν can be written as the sum of a unit matrix $\mathbf{I}$ and a matrix χ that requires the inversion of a $2R \times 2R$ matrix [18]

$$\phi_\nu(\mathbf{X}, \nabla_{\mathbf{X}}\mathcal{L}) = \mathbf{I} + \chi_\nu(\mathbf{X}, \nabla_{\mathbf{X}}\mathcal{L}) \tag{23}$$
$$\chi_\nu(\mathbf{X}, \nabla_{\mathbf{X}}\mathcal{L}) = -\nu\mathbf{A}(\mathbf{I} + \frac{\nu}{2}\mathbf{B}^T\mathbf{A})^{-1}\mathbf{B}^T \tag{24}$$
$$\mathbf{A} = [\nabla_{\mathbf{X}}\mathcal{L}, \mathbf{X}]$$
$$\mathbf{B} = [\mathbf{X}, -\nabla_{\mathbf{X}}\mathcal{L}].$$

With this transformation, the update (21) and (22) are also put in additive form corresponding to the weight update step defined in (11)

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \chi_\nu(\mathbf{U}^k, \nabla_{\mathbf{U}^k}\mathcal{L})\mathbf{U}^k \tag{25}$$
$$\mathbf{V}^{k+1} = \mathbf{V}^k + \chi_\nu(\mathbf{V}^k, \nabla_{\mathbf{V}^k}\mathcal{L})\mathbf{V}^k. \tag{26}$$

2) Singular Values: The update equations for the coefficients, or equivalently the singular values, can be calculated given the two transformations of the orthogonal matrices and the partial derivatives of the objective function to each component. Starting from the regular update equation (11) and the SVD notation in (8) at iteration $k+1$, an expression for the singular values update is obtained by equating both and solving for $\mathbf{S}^{k+1}$. Isolating the singular value matrix is done by using the orthogonality of both $\mathbf{U}$ and $\mathbf{V}$. This results in the expression given by the following equation:

$$\mathbf{S}^{k+1} = (\mathbf{U}^{k+1})^T \cdot [\mathbf{W}^k + \alpha\nabla_{\mathbf{W}}\mathcal{L}] \cdot \mathbf{V}^{k+1}. \tag{27}$$

The gradient with respect to the assembled matrix in terms of the component derivatives is given in Theorem 3.

Theorem 3: Derivative of objective function with respect to a matrix in the function of the derivatives with respect to its SVD components [22]

$$\nabla_{\mathbf{W}}\mathcal{L}|_{\nabla_{\mathbf{U},\mathbf{S},\mathbf{V}}\mathcal{L}} = \mathbf{D}\mathbf{V}^T + \mathbf{U}\mathbf{A}\mathbf{V}^T + \mathbf{U}\mathbf{S}\frac{\mathbf{B}+\mathbf{B}^T}{2}\mathbf{V}^T$$
$$\mathbf{A} = \nabla_{\mathbf{S}}\mathcal{L} - \mathbf{U}\mathbf{D}$$
$$\mathbf{B} = \mathbf{K} \circ [\mathbf{V}^T\nabla_{\mathbf{V}}\mathcal{L} - \mathbf{D}\mathbf{U}\mathbf{S}]$$
$$\mathbf{D} = [\nabla_{\mathbf{U}}\mathcal{L}\mathbf{S}^{-1}]_{\text{diag}}$$
$$\mathbf{K} = \begin{cases} \mathbf{S}_{ii} - \mathbf{S}_{jj}, & i \neq j \\ 0, & i = j. \end{cases} \tag{28}$$

Substituting (28) into (27) and using (23) gives the update equation for $\mathbf{S}^{k+1}$ only in terms of the components at iteration k and partial gradients. This is represented in the following equation:

$$\mathbf{S}^{k+1} = \left[\mathbf{I} + \Psi_{\mathbf{U}^k}^T\right] \cdot \mathbf{S}^k \cdot [\mathbf{I} + \Psi_{\mathbf{V}^k}] + \alpha\mathbf{Q}^k$$
$$\Psi_{\mathbf{X}} = (\mathbf{X})^T\chi_\nu(\mathbf{X})\mathbf{X}$$
$$\mathbf{Q}^k = \mathbf{U}^{k+1,T}\nabla_{\mathbf{W}}\mathcal{L}|_{\nabla_{\mathbf{U},\mathbf{S},\mathbf{V}}\mathcal{L}}\mathbf{V}^{k+1}. \tag{29}$$

Rewriting this as an additive equation gives the final equation [see (30)] that uses the diagonal components of an update matrix $\Delta_{\mathbf{S}}\mathcal{L}$

$$\mathbf{S}^{k+1} = \mathbf{S}^k + \Delta_{\mathbf{S}}\mathcal{L}$$
$$\Delta_{\mathbf{S}}\mathcal{L} = \left[\Psi_{\mathbf{U}^k}^T\mathbf{S}^k + (\mathbf{S}^k + \Psi_{\mathbf{U}^k}^T\mathbf{S}^k)\Psi_{\mathbf{V}^k} + \alpha\mathbf{Q}^k\right]_{\text{diag}}. \tag{30}$$

C. Initialization

Given the update equations from Section III-B, each component also needs proper initialization as convergence is strongly dependent on initial weights [23]. Neural network weights are initialized by randomly sampling each element from a specific distribution. Two distributions, namely, the Gaussian and uniform distribution, are most commonly used as initialization distributions. In terms of the components in the proposed decomposition, both $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. Any set of vectors can be made orthonormal using the Gramm–Schmidt procedure [24]. Using this procedure, R random vectors for both $\mathbf{U}$ and $\mathbf{V}$ are constructed. Since both these orthogonal matrices do not determine any distribution of the resulting composition matrix, the initialization of the singular value vector is responsible for the weight distribution. As such, a specific initialization for the R singular values is needed for both the Gaussian and uniform cases with an arbitrary variance.

1) Normal: Initialization for the Gaussian case corresponds to sampling the initial weights from the following distribution:

$$p(\mathbf{W}) = \mathcal{N}(0, \sigma_{\mathbf{W}}). \tag{31}$$

Let us now consider the distribution of the singular values contained in $\mathbf{S}$. According to Fisher–Hsu–Girshick–Roy [25], the singular values of a random Gaussian matrix have a marginal distribution described in Theorem 4.

Theorem 4: The marginal distribution for the unordered singular values λ of $\mathbf{X} \sim \mathcal{N}(0, 1) \in \mathbb{R}^{M\times N}$ is given by

$$p(\lambda)_{r,t} = \frac{1}{t}\sum_{k=0}^{t-1}\frac{k!}{(k+r-t)!}\left[L_k^{r-t}(\lambda)\right]^2\lambda^{r-t}\lambda^{-s} \tag{32}$$
$$L_k^{r-t}(s) = \frac{e^s}{k!s^{r-t}}\frac{d^k}{ds^k}(e^{-s}s^{n+r-t}). \tag{33}$$

Theorem 4 makes use of the Laguerre polynomials (33). These polynomials are orthogonal with respect to the weighting function $e^s s^{r-t}$, as portrayed by the following equation [26]:

$$\int_0^\infty e^{-x}x^k\left[L_n^k(x)\right]^2 dx = \frac{\Gamma(n+k+1)}{n!}. \tag{34}$$

This allows us to define a set of orthonormal functions as in (35), while the probability distribution takes the form of (36)

$$\phi_k^{r-t}(s) = \left[\frac{(k+r-t)!}{k!}\right]^{\frac{1}{2}} L_k^{r-t}(s) s^{\frac{r-t}{2}} e^{-\frac{s}{2}} \tag{35}$$

$$p(s)_{r,t} = \frac{1}{t}\sum_{k=0}^{t-1}\left[\phi_k^{r-t}(s)\right]^2. \tag{36}$$

This also shows that p_s is a probability distribution since its integral is one. This can be seen directly from the property in (34). To extend this to general variance with zero mean, a change of variables is done

$$p_\sigma(s)_{r,t} = |\sigma^2| p(s)_{r,t}. \tag{37}$$

Initialization can now be performed by sampling values from this distribution and putting them on the diagonal of $\mathbf{S}$.

2) Uniform: Initialization for the uniform case corresponds to sampling the initial weights from the following distribution:

$$p(\mathbf{W}) = \mathcal{U}(-a, a). \tag{38}$$

In a similar fashion as Section III-C1, a statement about the distribution of the singular values of this random matrix is needed. Theorem 5 states the limiting distribution of the singular values $\{\sigma_i^2\}$ for a uniform distribution given a symmetric interval and general variance.

Theorem 5: Given $Y \sim \mathcal{U}(-(\sigma/\sqrt{3N}), (\sigma/\sqrt{3N})) \in \mathbb{R}^{M\times N}$, in the limit $N, M \to \infty$ at fixed $M/N = r \in (0, +\infty]$ the singular values of Y have the Marcenko–Pastur distribution [see (39)][2]

$$p(\lambda) = \frac{1}{2\pi\lambda r\sigma^2}\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}$$
$$\lambda_\pm = \sigma^2(1 \pm \sqrt{r})^2. \tag{39}$$

This theorem, however, is not an exact statement as Theorem 4, but rather a limiting case. To verify the validity of the formula, the χ^2-statistic, given in (40), for histogram comparison [27] is calculated as an estimation of the error. The χ^2-statistic represents a measure for the deviation from the expected amount of counts per bin. c_i^0 and c_i^1 are the bin counts for each histogram using identically binned histograms

$$\chi(c^0, c^1) = 2\sum_i \frac{\left(c_i^0 - c_i^1\right)^2}{c_i^0 + c_i^1}. \tag{40}$$

Table I gives the deviations over 50k sampled singular values for different sizes of matrices, where N and M are described by the columns and rows, respectively. For smaller values of matrix sizes, there is some discrepancy, but negligible considering that the maximal deviation is $\pm 1\%$ of the sampled values. Furthermore, these smaller matrix sizes are practically irrelevant for neural networks. Based on these results, we can safely adopt the formula for uniform distributions for initialization of the singular values.

[2]Proof in Appendix A-B.

TABLE I

χ^2-STATISTIC OVER 50k SAMPLES FOR GAUSSIAN (WHITE) AND UNIFORM (GRAY) INITIALIZATION FOR DIFFERENT MATRIX SIZES

N \ M	16	32	64	128	256	512
16	5.12	12.89	31.09	81.94	207.38	593.92
	3.46	6.41	13.46	32.86	85.81	227.92
32	3.71	3.95	10.83	25.79	70.11	211.76
	2.66	2.92	5.19	12.76	28.73	74.69
64	2.38	2.66	3.66	10.0	27.65	75.95
	1.86	2.18	2.57	5.22	11.08	29.99
128	1.59	1.66	2.12	3.31	9.1	27.03
	1.27	1.55	1.84	2.51	4.83	10.77
256	1.03	1.08	1.26	1.84	3.3	9.11
	0.93	0.93	1.42	1.75	2.23	4.46
512	0.96	0.78	0.87	1.06	1.6	13.11
	0.7	0.78	0.93	1.2	1.47	2.07

D. Optimizer Dependence

The update equations (25), (26), and (30) all coincide with regular SGD by construction. Current state-of-the-art neural networks, however, are commonly trained using various methods of acceleration for stochastic optimization [28], [29]. The two main components for acceleration, i.e., momentum and adaptive learning rates, are discussed from the aspect of both the basis matrices and the coefficients in the decomposition in the following. Primarily, the effect of acceleration on the orthogonality, when optimizing over the Stiefel manifold, is examined. The necessary changes to a naive implementation are made and added to the training routine.

1) Momentum: One way of accelerating optimization algorithms is adding momentum to each weight update, as portrayed by (41) and (42). Momentum is named after the phenomenon from physics and is combined with interpretation of optimization as a ball rolling down a loss surface, which keeps its previous momentum after a change in direction. This translates into keeping the gradient of the previous step partially in mind when optimizing the current iteration

$$w^{k+1} = w^k + \alpha \cdot \theta^k \tag{41}$$

$$\theta^k = \beta \cdot \theta^{k-1} + \nabla_{w^k}\mathcal{L}| \quad \theta^0 = 0. \tag{42}$$

When considering momentum for (25) and (26), the orthogonality of $\mathbf{U}$ and $\mathbf{V}$ deteriorates with each iteration. Different implementations, such as moving average momentum implemented in the ADAM optimizer, that scale each new gradient term in (42) by a factor of $1-\beta$, are technically the identical and suffer from the same drawback. In light of Theorem 3 and Fig. 1, this problem can be solved by applying the momentum step in (42) with the gradient of which the action is calculated. Due to the linearity of this action, this is equivalent to applying the momentum step in the tangent space of the manifold, as in [7], and projecting afterward. To stay consistent, this same method is applied to the gradients of the singular values.

2) Adaptive Learning Rate: Any gradient update as given in (11) consists of a gradient part and a step size or learning rate. By directly changing the latter, the speed of learning can be accelerated or slowed down. Formally, this is described by

making the learning rate iteration dependent ($\alpha \rightarrow \alpha^k$) as in the following equation:

$$w^{k+1} = w^k + \alpha^k \cdot \nabla_{w^k} \mathcal{L}. \tag{43}$$

Similar to momentum, running averages of gradient norm are kept that scale the learning rate as portrayed in (44). The following equations represent the variable learning rate scheme used in the ADAM optimization algorithm:

$$\begin{aligned} \alpha^k &= \frac{\alpha}{\sqrt{v^k} + \epsilon} \\ v^k &= \gamma\, v^{k-1} + (1-\gamma)\|\nabla_{w^k}\mathcal{L}\|^2| \quad v^0 = 0. \end{aligned} \tag{44}$$

When considering adaptive learning rates, the optimization on the Stiefel manifold runs into some problems. Since the calculation of χ_ν requires a scalar value for ν, the adaptive learning rate has to be addressed before calculation of χ_ν. Looking at (44), this adaptive learning rate is a scalar scaled inversely by a matrix, which is done elementwise. By applying this elementwise operation to the gradient or moment matrix, as in (41) and (11), respectively, the learning returns to being a constant scalar and is compatible with the Cayley transform calculation. This adaptation can be interpreted as scaling of the gradient or momentum for each matrix entry and is applicable to all adaptive learning rate schedules that allow extraction of a scalar. It is here where reintroduction of the omitted line search from Section III-B1 is also a possibility. However, as mentioned before, we did not follow this approach for computational reasons and used standard acceleration schemes from the state-of-the-art stochastic optimization. In line with the treatment of the singular value gradients regarding momentum updates, this way of implementing the adaptive learning rate is also implemented for the singular value updates.

IV. Experiments

In this section, all experiments comparing our proposed method to the standard interpretation of weight matrices are described. Everything was performed using two Tesla V100-SXM3-32GB GPU and eight Intel Xeon Platinum 8168 CPU @ 2.70 GHz CPU cores. Each experiment is a specific implementation of the pseudocode described in Algorithm 1 using Tensorflow 2.2.0. Initialization used for all experiments using either the standard Tensorflow library or the corresponding initialization from Section III-C. Implementations can be found in the open-source code.[3]

Algorithm 1 Accelerated SVD Optimization Scheme

Input: $\{\mathbf{U}^{k-1}, \mathbf{S}^{k-1}, \mathbf{V}^{k-1}\}$
for $\mathbf{X}^{k-1}$ in $\{\mathbf{U}^{k-1}, \mathbf{S}^{k-1}, \mathbf{V}^{k-1}\}$ **do**:
  $\nabla_{\mathbf{X}^{k-1}}\mathcal{L} \leftarrow GetGradient(\mathbf{X}^{k-1})$
  $\theta_{\mathbf{X}^k} \leftarrow Acceleration(\nabla_{\mathbf{X}^{k-1}}\mathcal{L}, \theta_{\mathbf{X}^{k-1}})$
for $\mathbf{X}^{k-1}$ in $\{\mathbf{U}^{k-1}, \mathbf{V}^{k-1}\}$ **do**:
  $\mathbf{X}^k \leftarrow \mathbf{X}^{k-1} + \chi_\nu(\mathbf{X}^{k-1}, \theta_{\mathbf{X}^k})$
$\mathbf{S}^k \leftarrow \mathbf{S}^{k-1} + \Delta\mathbf{S}_{\nu,\alpha}(\{\mathbf{X}^{k-1}\}, \{\theta_{\mathbf{X}^k}\})$
return $\mathbf{U}^k$, $\mathbf{S}^k$, $\mathbf{V}^k$

[3]https://github.com/predict-idlab/svd-kernels

The performed experiments are divided into three sections. First, regular feedforward networks are evaluated for different matrix sizes and ranks using optimization with SGD, SGD with momentum, and ADAM as described in Section III-D. Second, the proposed method is used to substitute fully connected layers in residual network (ResNets) architectures. Finally, the transformer architecture is considered in which each weight matrix is substituted for its decomposed variant. This order of experiments is such that the properties of the decomposed weight matrices can be observed and, consequently, their performance for relevant architectures evaluated. For all experiments, visual evaluations are done, which contains both the learning and accuracy curves during training, tracking of the orthogonality conditioning number ($\kappa_{\text{orthogonality}}$), which reflects how orthogonal the $\mathbf{U}$ and $\mathbf{V}$ components are at every step, given by (45) and a tracker for the conditioning number ($\kappa_{\text{conditioning}}$) as defined in (10). All figures, both discussed in the following and additional ones, can be found in Appendix B

$$\kappa(\mathbf{x}) = \sqrt{||\mathbf{x}^T \cdot \mathbf{x} - \mathbf{I}||}. \tag{45}$$

A. Feedforward

The first set of experiments compares regular and decomposed weight matrices using feedforward networks. For this purpose, a feature-based dataset is selected, the cleveland-heart-disease dataset [30] with 32 input features. A single hidden layer neural network with one sigmoidal output node for binary classification is evaluated. Binary cross entropy is used as loss function and binary accuracy as a validation metric. A comparison for different ranks and hidden layer sizes is done for each of the three optimization schemes mentioned before. The results on a 20% validation split are given in Tables II–IV for standard SGD, SGD with momentum, and ADAM, respectively. All of these values are averaged over 25 training runs, and for each matrix size, the best performing architecture has been highlighted.

Tables V and VI contain the amount of parameters and averaged training times for each model, respectively. Visual performance evaluation is included for three different parameter combinations. First, the 32 unit matrix size with rank 32 is selected for its high regular SGD performance, and being the only matrix size, the regular approach outperforms its decomposed variant. Second, for the maximum size matrix 512, which reaches maximal overall methods, rank 24 variant is selected since it is the best performing. Finally, matrix size 128 with rank 16 is added for intermediate comparison.

Comparing different optimizer performances in Tables II–IV shows that, unlike regular weight matrices, the decomposed matrices are a lot more independent of the chosen optimization scheme. This observation is attributed to the more directed search granted by using the Stieffel manifold for optimization of the orthogonal components of the decomposition. Eventually, both approaches reach maximal performance using the ADAM optimization scheme. From the corresponding results, presented in Table IV, the decomposed weight matrices outperform their regular counterparts consistently except for

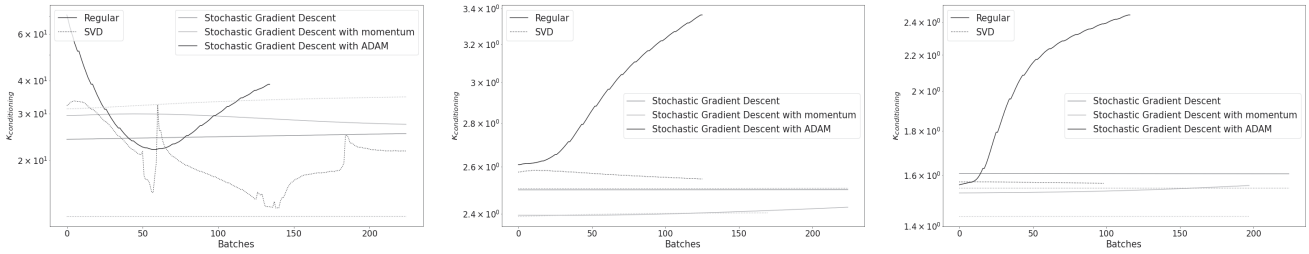


Fig. 2. Conditioning curves for the selected feedforward models. Given are the models with (a) 32 units and rank 32, (b) 128 units and rank 16, and (c) 512 units and rank 24.

TABLE II
FEEDFORWARD MODEL ACCURACY FOR SGD

R \ N	32	64	128	256	512
8	50.16	56.51	59.37	**77.46**	77.14
16	53.33	67.30	70.48	73.33	**79.68**
24	60.95	**69.21**	**74.60**	76.83	79.37
32	**62.54**	65.08	73.02	74.92	77.14
Full	53.41	55.40	53.33	63.02	64.05

TABLE III
FEEDFORWARD MODEL ACCURACY FOR SGD WITH MOMENTUM

R \ N	32	64	128	256	512
8	75.24	77.14	77.46	79.37	77.78
16	**79.37**	78.41	80.00	79.37	80.63
24	77.14	79.37	79.68	80.32	80.63
32	75.56	**80.32**	**82.54**	**80.63**	**80.95**
Full	75.71	76.98	78.41	79.21	78.81

TABLE IV
FEEDFORWARD MODEL ACCURACY FOR ADAM

R \ N	32	64	128	256	512
8	78.73	**81.59**	**81.27**	81.27	80.32
16	79.37	80.63	**81.27**	80.63	80.32
24	78.41	81.27	80.32	80.63	**82.22**
32	79.05	80.63	80.00	**81.59**	81.90
Full	**80.63**	81.19	80.56	80.79	80.87

TABLE V
FEEDFORWARD MODEL PARAMETER COMPARISON

R \ N	32	64	128	256	512
8	569	889	1529	2809	5369
16	1073	1649	2801	5105	9713
24	1577	2409	4073	7401	14057
32	2081	3169	5345	9697	18401
Full	1025	2049	4097	8193	16385

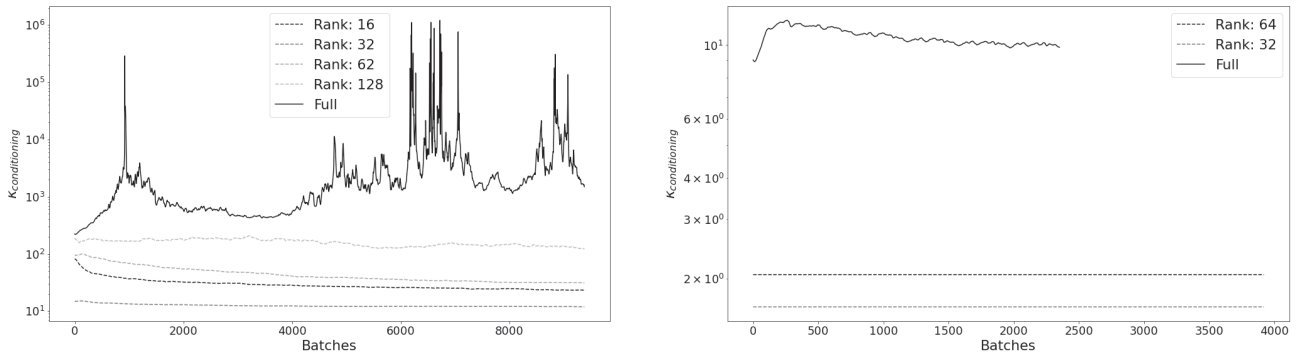
TABLE VI
FEEDFORWARD MODEL TRAIN TIME COMPARISON (s/epoch)

R \ N	32	64	128	256	512
8	1.901	1.946	1.942	1.897	1.904
16	1.913	1.867	1.935	2.013	1.968
24	1.938	1.979	2.015	1.878	1.940
32	1.949	1.956	2.004	2.007	1.991
Full	1.168	1.199	1.177	1.178	1.190

the square matrix at 32 units. This observation seems in line with the assumption that decreasing the rank and directly learning well-conditioned matrices avoid overparameterization. This more efficient parameterization is also reflected in the number of parameters given in Table V. Table VI, however, represents the downside of the method since, at each rank, the computational time goes up significantly. When comparing the conditioning number tracker given in Fig. 2(a) with those in Fig. 2(b) and (c), this suspicion is reaffirmed. For the regular square matrix, the conditioning number does not grow out of bounds with respect to the decomposed ones. In contrast, for the more rectangular matrices, the conditioning number keeps rising throughout training, while the decomposed variants keep themselves well-conditioned. This ill-conditioned behavior likely negates some of the benefits of the increased hidden dimension. By using the decomposed matrices, the size of the hidden dimension can be enlarged sufficiently without these side effects, leading to increased performance. Both the 256 units/rank 32 and 512 units/rank 24 matrices significantly outperform the regular implementation at all sizes for the validation accuracy.

B. Residual Networks

Following the previous results on feedforward networks, the decomposition layer is also introduced in more complicated state-of-the-art architectures, more specifically ResNets for image recognition [31]. Both the MNIST and CIFAR10 benchmark datasets are used for evaluation. For MNIST, a single residual block with 256 filters followed by a single feedforward layer of 256 units is used, and for CIFAR10, a triple residual block architecture with 64, 128, and 256 filters followed by a 1000-unit feedforward layer is used. Each residual block is separated by a 2 times 2 average-pooling operation. It is both these feedforward layers that are evaluated with the regular and decomposed method. In line with the



(a) (b)

Fig. 3. Conditioning curves for (a) MNIST and (b) CIFAR10.

TABLE VII
PERFORMANCE COMPARISON FOR DIFFERENT RANKS OF THE 256 UNIT FEEDFORWARD LAYER OF THE RESNET ARCHITECTURE ON MNIST

Rank	Time (Batch/s)	Accuracy (%) Train	Test	Parameters
16	25	98.68	98.47	$\approx 180k$
32	24	98.71	98.51	$\approx 188k$
64	21	**99.17**	98.86	$\approx 203k$
128	20	98.88	**98.88**	$\approx 237k$
Full	28	99.05	98.74	$\approx 237k$

TABLE VIII
PERFORMANCE COMPARISON FOR DIFFERENT RANKS OF THE 256 UNIT FEEDFORWARD LAYER OF THE RESNET ARCHITECTURE ON CIFAR10 EVALUATED AT 20 EPOCHS OF TRAINING

Rank	Time (Batch/s)	Accuracy (%) Train	Test	Parameters
32	14	94.97	83.83	$\approx 267k$
64	16	93.19	83.74	$\approx 271k$
128	17	88.73	83.33	$\approx 278k$
Full	18	**95.95**	**83.88**	$\approx 277k$

findings from Section IV-A, all optimization is done using the ADAM optimization scheme since this results in the best performance for both the decomposed layer and all other variables such as the convolutional layers. The initial learning rate used is 0.01 and is decreased to 0.001 after 20 epochs. L_2 regularization with coefficients 0.001 and 0.01 is used for the convolutional and feedforward layers, respectively. In the case of the decomposed variants, this regularization is imposed on the singular values. Tables VII and VIII summarize the performance comparison between multiple rank decomposed approaches and the full-rank regular approach for MNIST and CIFAR10, respectively. The corresponding learning, accuracy curves, orthogonality, and conditioning numbers are given in Appendix B-B.

Similar observations as in Section IV-A are seen in the MNIST evaluation. At smaller ranks corresponding to fewer parameters, the same performance as with full-rank matrices is achieved, and at a similar amount of parameters, rank 128, the decomposed matrix outperforms the regular weight matrix slightly. Furthermore, at all ranks, the discrepancy between train and test error is slightly smaller. This is indicative of better generalization properties that are linked to the conditioning number of the feedforward layer, as shown in Fig. 3(a). Here, it is clearly seen that the regular matrix implementation is much worse conditioned than the decomposed counterparts. From the training times, a growing discrepancy is seen as the rank of the decomposition grows. This is due to the number of matrix multiplications and inverse needed in the backpropagation step. For CIFAR10, however, no significant improvement in performance is observed. This seems

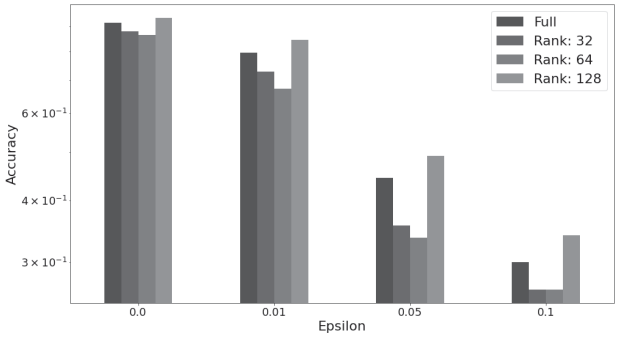


Fig. 4. Model accuracies for adversial examples generated using FSGM with different values of ϵ on a random 1000 image subset of the CIFAR10 test data.

reasonable since the amount of convolutional parameters greatly outnumbers the number of weight in the feedforward layer and is also reflected in no significant computational discrepancy. However, as presented in Fig. 3(b), the conditioning of these feedforward layers is still much better for the decomposed layers. Since ill-conditioned matrices lead to large output fluctuations, these better conditioned matrices have the potential of being more robust to noisy input. To verify this assumption, the models are evaluated on a set of adversial examples generated on the CIFAR10 dataset using the fast gradient sign method (FSGM) [32] with a parameter epsilon. This method adds a perturbation to the original image consisting of the gradient of the image with respect to the class label given a set of model parameters with θ as given in the

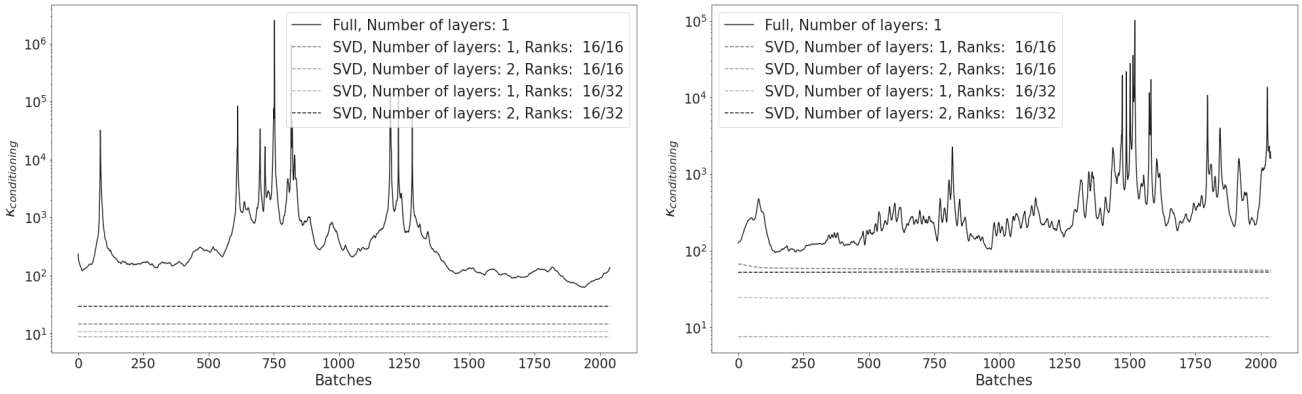


Fig. 5. Conditioning number tracking in the first encoder for different transformers. Each model has depth 64 and width 256 and the vocabulary size used is 1000 words. (a) First encoder, multihead attention query. (b) First encoder, feedforward layer.

following equation:

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \vec{y}, \theta). \tag{46}$$

Fig. 4 shows the results at different values for ϵ for 1000 adversial examples created from the CIFAR10 dataset. The rank 128 model outperforms the regular weight matrix consistently and resists the FSGM attacks much better. This is in line with the smaller discrepancy between train and test error for the rank 128 model, as observed in Table VIII. On the other hand, the lower rank models do not perform well on this test. This observation shows another potential downside, which is that when the rank is too small, the decomposed weight matrices do not have enough expressive capacity, seemingly leading to worse performance under perturbation. This reiterates the importance of having a correct R value.

C. Transformer

In the following set of experiments, the transformer architecture introduced by Vaswani *et al.* [33] is evaluated on the TED-Talks English to Portuguese translate dataset from the tensorflow datasets library. Standard accuracy for the predicted words is used as an evaluation metric. The transformer architecture is well suited for the change of regular to decomposed weight matrices since a single layer consists of a set of multihead attention and feedforward layers. The attention mechanism contains four linear weight multiplications and the feedforward layers each have one. These can all be converted to decomposed weight matrices. Both the feedforward and attention have their own size and rank. For the attention layers, the size is referred to as depth, and for the feedforward layers, the size is referred to as width. The full transformer architecture consists of an encoder and decoder that both have a set number of these composite layers. Each transformer is trained using the ADAM optimizer as described in Section III-D for optimal performance. Preprocessing is done using a standard word tokenizer with the number of words as a hyperparameter and the embeddings are trained from scratch for each implementation.

TABLE IX

COMPARISON OF TRANSFORMER PERFORMANCE FOR DIFFERENT RANKS (R) AND NUMBER OF LAYERS (L). EACH MODEL HAS DEPTH 64 AND WIDTH 256 AND THE VOCABULARY SIZE USED IS 1000 WORDS. THE ITALIC HIGHLIGHTED MODELS ARE THE TWO WITH QUASI IDENTICAL AMOUNT OF PARAMETERS, WHILE THE BOLD HIGHLIGHTING REFLECTS THE BEST PERFORMING ARCHITECTURE

Model	Time (Batch/s)	Accuracy (%) Train	Test	Parameters
REG, L: 1	8	98.92	98.87	$\approx 310k$
SVD, L: 1, R: 16/16	6	99.29	98.82	$\approx 241k$
SVD, L: 2, R: 16/16	4	99.32	98.88	$\approx 288k$
SVD, L: 1, R: 16/32	6	99.15	99.07	$\approx 251k$
SVD, L: 2, R: 16/32	4	**99.37**	**99.17**	$\approx 308k$

We compare a set of transformer architectures with depth 64 and width 256 using a vocabulary size of 1000 words. The architectures are summarized by a number of layers for both decoder and encoder described by the parameter L and a tuple of ranks for the depth and width, respectively, when using the decomposed weight matrices. At both values 16/16 and 16/32 for the depth rank and width rank, respectively, a double-layer encoder and decoder contain less parameters than the full matrix single-layer transformer. As such, both the single- and double-layer decomposed architectures are compared to the single-layer regular transformer. Table IX summarizes the results, while the corresponding learning and accuracy curve can be found in Appendix B-C.

For each decomposed transformer model, the training accuracy is higher than the regular model, while on the test set, only the rank 16/32 models outperform and the rank 16/16 models achieve a similar performance. At the same amount of parameters, the two layers decomposed with rank 16/32 model achieve a small but significant improvement over the regular weight matrix transformer, as indicated in bold in Table IX. It seems that, at these ranks, the separate layers retain enough expressive capacity, while the reduction in parameters allows for a second layer, leading to more complex modeling and eventually increased performance. Fig. 5 shows the conditioning of the query weight matrix in the multihead attention

and the following feedforward layer in the first encoder. The conditioning numbers behave similarly as in previous experiments, leading to better conditioned matrices. In terms of training times, a significant increase of 150% for the single layer and 200% for the double-layer decomposed transformer is seen and confirms the main downside of the decomposed method. This is due to the matrix inversion in (14) being calculated exactly and the large number of matrix multiplications during backpropagation. In [9], the Cayley transform is calculated iteratively, which reduces the accuracy in favor of computation time. Since the orthogonal matrices are stored instead of the inverse Cayley transformed skew-symmetric matrix, as proposed in [7], these computations are not present during inference. We accept the computational downside during training in favor of the upsides of increased performance, generalization, parameter efficiency, and fast inference times. Moreover, all the training time results used in both the ResNets as transformers are reported without the calculation of the SVD decomposition of the regular weight matrix at each iteration needed for the conditioning and orthogonality tracking. When access to the singular values is required, the decomposed formulation has significant advantages.

V. Conclusion

In this article, a different way of approaching weight matrices in neural networks is proposed based on the observation that the set of real matrices is a vector space under addition and multiplication. As such, any matrix can be described as a fitting linear combination of basis matrices. Formulating each basis matrix as a Kronecker product of two vectors makes the amount of trainable parameters feasible and unifies this approach with the SVD of the weight matrix. Using the Cayley transform and a corresponding descent curve on the Stiefel manifold, we constructed update equations for the orthogonal matrices of which the basis matrices are composed. This constraint on the possible values of the weights allows for a more directed search, leading to improved performance and generalization. More stable performance over different optimization methods compared to the regular weight matrix approach is observed. A first direct consequence of this formulation is the control over the rank of the weight matrix. Supposedly, each linear transformation could be sufficiently described or approximated by a rank-deficient matrix. A second consequence is that direct access to the singular values allows much better conditioning of the matrices by regularizing the singular values and consequently the conditioning number κ. Both these properties are observed in experiments, allowing for more parameter-efficient formulations of neural networks without loss of performance. Furthermore, special routines for both uniform and Gaussian initialization are derived, which allows the state-of-the-art initialization for each component in the R-dimensional decomposition.

In future work, other possibilities for the construction of basis sets can be considered, keeping in mind the constraint on the number of parameters per layer. Furthermore, the idea that the relation blocks represented by the basis components could be reused in different layers can be entertained, thus leading to a reduction in the number of parameters. Not only the basis components but also the coefficients and the direct access to them open up possibilities. Current neural networks have static architectures and static weights for each sample. By allowing the coefficients to be a direct function of the input or, preferably, of the context of the input, dynamic networks that adapt better to specific inputs could be created.

We are convinced that the proposition of weight kernels as linear combinations of basis components gives much more flexibility and will open avenues to more data-efficient and better learning.

References

[1] S. Chakraverty, D. M. Sahoo, and N. R. Mahato, "McCulloch–Pitts neural network model," in *Concepts of Soft Computing*. New York, NY, USA: Springer, 2019, pp. 167–173.

[2] B. Widrow and M. A. Lehr, "Perceptrons, adalines, and backpropagation," in *Arbib*, vol. 4, 1st ed., Jun. 1995, pp. 719–724.

[3] R. Bhatia, *Matrix Analysis*, vol. 169. New York, NY, USA: Springer, 2013.

[4] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Aug. 2011.

[5] B. Dumitrescu and P. Irofti, *Dictionary Learning Algorithms and Applications*. New York, NY, USA: Springer, 2018.

[6] J. Zhang, Q. Lei, and I. S. Dhillon, "Stabilizing gradients for deep neural networks via efficient SVD parameterization," 2018, *arXiv:1803.09327*.

[7] K. Helfrich, D. Willmott, and Q. Ye, "Orthogonal recurrent neural networks with scaled Cayley transform," 2017, *arXiv:1707.09520*.

[8] Z. Mhammedi, A. Hellicar, A. Rahman, and J. Bailey, "Efficient orthogonal parametrisation of recurrent neural networks using householder reflections," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2401–2409.

[9] J. Li, L. Fuxin, and S. Todorovic, "Efficient Riemannian optimization on the Stiefel manifold via the Cayley transform," 2020, *arXiv:2002.01113*.

[10] X. Zhu, "A Riemannian conjugate gradient method for optimization on the Stiefel manifold," *Comput. Optim. Appl.*, vol. 67, no. 1, pp. 73–110, May 2017.

[11] Y. Nishimori and S. Akaho, "Learning algorithms utilizing quasi-geodesic flows on the Stiefel manifold," *Neurocomputing*, vol. 67, pp. 106–135, Aug. 2005.

[12] S. Wisdom, T. Powers, J. R. Hershey, J. Le Roux, and L. Atlas, "Full-capacity unitary recurrent neural networks," 2016, *arXiv:1611.00035*.

[13] K. Jia, D. Tao, S. Gao, and X. Xu, "Improving training of deep neural networks via singular value bounding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4344–4352.

[14] S. T. Wauthier, O. Çatal, C. De Boom, T. Verbelen, and B. Dhoedt, "Sleep: Model reduction in deep active inference," in *Proc. Int. Workshop Act. Inference*. New York, NY, USA: Springer, 2020, pp. 72–83.

[15] H. Zhang and F. Ding, "On the Kronecker products and their applications," *J. Appl. Math.*, vol. 2013, pp. 1–7, Jun. 2013.

[16] C. Jose, M. Cisse, and F. Fleuret, "Kronecker recurrent units," 2017, *arXiv:1705.10142*.

[17] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*. New York, NY, USA: Springer, 2010, pp. 177–186.

[18] H. D. Tagare, *Notes on Optimization on Stiefel Manifolds*. New Haven, CT, USA: Yale Univ., 2011, p. 33.

[19] W. Sun and Y.-X. Yuan, *Optimization Theory and Methods: Nonlinear Programming*, vol. 1. Berlin, Germany: Springer, 2006.

[20] J. Nocedal and S. Wright, *Numerical Optimization*. Berlin, Germany: Springer, 2006.

[21] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien, "Painless stochastic gradient: Interpolation, line-search, and convergence rates," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 3732–3745.

[22] C. Ionescu, O. Vantzos, and C. Sminchisescu, "Training deep networks with structured layers by matrix backpropagation," 2015, *arXiv:1509.07838*.

[23] S. K. Kumar, "On weight initialization in deep neural networks," 2017, *arXiv:1704.08863*.

[24] W. Hoffmann, "Iterative algorithms for Gram–Schmidt orthogonalization," *Computing*, vol. 41, no. 4, pp. 335–348, Dec. 1989.
[25] A. M. Tulino, S. Verdú, and S. Verdu, *Random Matrix Theory and Wireless Communications*. Norwell, MA, USA: Now, 2004.
[26] A. D. Poularikas, *The Handbook of Formulas and Tables for Signal Processing*. Boca Raton, FL, USA: CRC Press, 1999.
[27] W. G. Cochran, "The χ^2 test of goodness of fit," *Ann. Math. Statist.*, vol. 23, pp. 315–345, Sep. 1952.
[28] Z. Allen-Zhu, "Katyusha: The first direct acceleration of stochastic gradient methods," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 8194–8244, Jan. 2017.
[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[30] K. U. Rani, "Analysis of heart diseases dataset using neural network approach," 2011, *arXiv:1110.2626*.
[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[32] C. Zuo, "Regularization effect of fast gradient sign method and its generalization," 2018, *arXiv:1810.11711*.
[33] A. Vaswani *et al.*, "Attention is all you need," 2017, *arXiv:1706.03762*.
[34] P. Yaskov, "A short proof of the Marchenko–Pastur theorem," *Comp. Rendus Mathematique*, vol. 354, no. 3, pp. 319–322, Mar. 2016.
[35] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," *Tech. Univ. Denmark*, vol. 7, no. 15, pp. 1–72, Nov. 2008.

David Vander Mijnsbrugge received the M.Sc. degree in engineering physics from Ghent University, Ghent, Belgium, in 2019, where he is currently pursuing the Ph.D. degree in computer science at the IDLab, with a focus on context-aware machine learning in eHealth.

During this master degree, he also worked as a part-time NLP engineer. The focus of this work is adaptation of deep learning for more dynamic behavior in different contexts and integration with knowledge systems.

Femke Ongenae received the Ph.D. degree in computer science from Ghent University, Ghent, Belgium, August 2013, pertaining to knowledge discovery and management for eHealth applications by using ontologies and semantic reasoning.

She has been a part-time Assistant Professor at the IDLab, Ghent University, since October 2019. She is currently a full-time Research Manager and a Senior Scientist at the imec research hub, Ghent, for nanotechnologies and digital technologies. During this time, she worked on several eCare projects to improve the continuous care of patients in institutionalized care settings. As a Professor at Ghent University, she is the part of PREDICT and KNOWS research teams. She performs research into expressive semantic stream and distributed reasoning; the incorporation of expert knowledge in data analytics algorithms; hybrid artificial intelligence (AI), fusing semantic models, and machine learning; and explainable AI by leveraging knowledge graphs. She is also particularly interested in methodologies for capturing domain knowledge from experts and using this knowledge to optimize intelligent agents and the way we interact with them. This research is mainly applied to the domains of predictive healthcare and industry 4.0 in order to realize context-aware and personalized decision support systems.

Sofie Van Hoecke received the M.Sc. and Ph.D. degrees in computer science engineering from Ghent University, Ghent, Belgium, in 2003 and 2009, respectively.

Since 2017, she has been an Associate Professor with the Internet Technology and Data Science Laboratory, Ghent University–imec, Ghent. Her research interests include the study and development of hybrid machine learning solutions combining machine learning with semantics, expert knowledge, and/or physical knowledge, with applications in predictive maintenance and predictive healthcare.