Dimensionality Reduction in Deep Learning via Kronecker Multi-layer Architectures*

Jarom D. Hogue† Robert M. Kirby‡ Akil Narayan§

April 12, 2022

Abstract

Deep learning using neural networks is an effective technique for generating models of complex data. However, training such models can be expensive when networks have large model capacity resulting from a large number of layers and nodes. For training in such a computationally prohibitive regime, dimensionality reduction techniques ease the computational burden, and allow implementations of more robust networks. We propose a novel type of such dimensionality reduction via a new deep learning architecture based on fast matrix multiplication of a Kronecker product decomposition; in particular our network construction can be viewed as a Kronecker product-induced sparsification of an "extended" fully connected network. Analysis and practical examples show that this architecture allows a neural network to be trained and implemented with a significant reduction in computational time and resources, while achieving a similar error level compared to a traditional feedforward neural network.

1. Introduction Statistical learning using deep neural networks has achieved impressive results in building models for prediction, summarization, and classification of large data sets [7]. The capacity of such models is dictated by the depth and width (number of layers and nodes, respectively) of the neural network, but such high-capacity networks impose a nontrivial computational burden during training. In such regimes, dimensionality reduction may be implemented in some form to reduce the computational burden. While this may take several forms, of particular interest is accelerating training of neural networks. One of the computational burdens that arise for high-capacity networks during training is the cost of forward- and back-propagation, amounting to the cost of evaluation of the network and the cost of implementing the computational graph corresponding to the chain rule for differentiation, respectively. In this paper, we analogize this problem to that of matrix multiplication: Matrix-vector multiplication for large matrices can be

*This research was sponsored by ARL under Cooperative Agreement Number W911NF-12-2-0023. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ARL or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. The first and third authors are partially supported by NSF DMS-1848508 and AFOSR FA9550-20-1-0338.

†Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT(jdhogue@sci.utah.edu).

‡Scientific Computing and Imaging Institute and School of Computing, University of Utah, Salt Lake City, UT(kirby@cs.utah.edu).

§Scientific Computing and Imaging Institute and Department of Mathematics, University of Utah, Salt Lake City, UT(akil@sci.utah.edu).

expensive, but is much more efficient if certain structural properties of matrices exist that can be computationally exploited. In particular we exploit the fact that the Kronecker product [18] provides a low dimensional representation of a large matrix, and use a corresponding implementation of this dimensionality reduction for deep learning.

Our approach aims to gain computational efficiency by imparting a Kronecker product structure on the *architecture* of a neural network. Ultimately we aim to accelerate training of neural networks without degrading predictive accuracy. We implement a "dual layer" approach that is inspired by the structure of a Kronecker product and show that this procedure can significantly reduce computational time for both forward computation and back-propagation, when the network size/capacity is relatively large, compared to a fully connected feedforward neural network. We also show in such cases that we can maintain or even improve accuracy when training on several practical examples.

In short, the contributions of this paper are as follows: (i) We propose a new Kronecker product-inspired deep learning architecture, the Kronecker Dual Layer (KDL), that exploits simplification of arithmetic operations in Kronecker products for matrix multiplication to effect acceleration in both forward- and back-propagation phases of learning; (ii) we provide proof-of-concept theoretical analysis suggesting when a KDL network can be expected to perform well compared to fully connected networks; (iii) we demonstrate the practical effectiveness of KDL architectures on real-world datasets through several test examples.

1.1. Related work The Kronecker product has already been incorporated in several areas within the deep learning framework: (i) In [14, 19] the authors apply a Kronecker product decomposition (KPD) to decompose weight matrices of a trained network, although this typically requires a large number of terms for acceptable accuracy and is thus of limited applicability; (ii) a generalized KPD is extended to multi-dimensional tensors in [8] to reduce the number of weight parameters and computational complexity in convolutional neural networks; (iii) the Kronecker product has been shown as a viable method to reduce the computational time for back-propagation via an approximate inverse of the Fisher information matrix, [13], providing a means to increase decay rate in the loss; (iv) a "Kronecker neural network" in [10], has been established to implement adaptive activation functions in order to avoid local minima while training. We emphasize that our approach is distinct from these methods, as we fundamentally alter network architecture in an attempt to accelerate training.

In addition to the alternative uses of the Kronecker product mentioned above, there are other methods that seek to reduce the nodes and/or connections of a trained network in order to reduce the computational burden of training or prediction. Dropout, see e.g. [9, 11, 16], randomly ignores nodes or connections with a set probability when training, and has the added benefit of reducing co-adaptation of features. Pruning, see e.g. [2, 20], on the other hand, seeks to force weights with a minimal impact to zero, thereby increasing sparsity within the trained network. Although these methods share the same broad goal as this work, our KDL approach splits layers based on a Kronecker product to form a new architecture, and is a novel means of reducing the required computational resources while maintaining accuracy.

1.2. Notation A plain lowercase letter v will denote either a scalar or a function, a bold lowercase letter $\boldsymbol{v}$ will represent a vector, and a plain uppercase letter V will represent a matrix. Subscripts will denote indices within a vector or matrix, with a colon denoting MATLAB-style slicing of the full range of indices. Parenthesis in the superscript will denote variations based on layer, terms

in a summand, and so forth, with multiple such designations separated by commas. In addition, matrices may be numbered with a single digit in the subscript, in which case indices will be noted in parenthesis following the single digit in the subscript.

2. Fully Connected Network An artificial neural network is a function defined by a series of compositions and can be identified by an activation function $\phi : \mathbb{R} \to \mathbb{R}$ and a collection of weight matrices and bias vectors. With n_1 and n_L the input and output dimensions of the map, respectively, a *fully connected neural network* (FNN) mapping input $\boldsymbol{x} \in \mathbb{R}^{n_1}$ to $\boldsymbol{y} \in \mathbb{R}^{n_L}$ can be defined as,

$$\boldsymbol{y} = \boldsymbol{a}^{(L)}, \qquad \boldsymbol{a}^{(\ell+1)} = \left(\phi \circ \tilde{h}_{W^{(\ell+1)},\boldsymbol{b}^{(\ell+1)}}\right)\left(\boldsymbol{a}^{(\ell)}\right), \ \ \ell = 1, \ldots, L-1, \tag{1}$$

where $\boldsymbol{a}^{(1)} = \boldsymbol{x}$, $L \in \mathbb{N}$ is the number of layers of the network, and $\tilde{h}_{W,\boldsymbol{b}}$ is an affine map defined through its weight matrix $W \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ and bias vector $\boldsymbol{b} \in \mathbb{R}^{n_{\ell+1}}$,

$$\tilde{h}_{W,\boldsymbol{b}} : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_{\ell+1}}, \qquad \tilde{h}_{\{W,\boldsymbol{b}\}}(\boldsymbol{a}) = W\boldsymbol{a} + \boldsymbol{b}.$$

In (1), ϕ operating on vectors is defined componentwise. With our notation, the $L-2$ intermediate stages $\left\{\boldsymbol{a}^{(\ell)}\right\}_{\ell=2}^{L-1}$ are *hidden layers*, and each component of $\boldsymbol{a}^{(\ell)}$ is a *node*. We let n_ℓ, $\ell \in [L]$, denote the number of units in layer ℓ, so that $\boldsymbol{a}^{(\ell)} \in \mathbb{R}^{n_\ell}$. Using the notation,

$$\begin{aligned}\tilde{\theta} &\coloneqq \left\{W^{(2)}, \boldsymbol{b}^{(2)}, \ldots, W^{(L)}, \boldsymbol{b}^{(L)}\right\} \text{ with} \\ \tilde{\theta}^{(\ell)} &\coloneqq \left\{W^{(\ell)}, \boldsymbol{b}^{(\ell)}\right\},\end{aligned}$$

the FNN input-to-output map is then,

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}; \tilde{\theta}) = \left(\phi \circ \tilde{h}_{\tilde{\theta}^{(L)}} \circ \phi \circ \tilde{h}_{\tilde{\theta}^{(L-1)}} \cdots \phi \circ \tilde{h}_{\tilde{\theta}^{(2)}}\right)(\boldsymbol{x}), \qquad \tilde{h}_{\tilde{\theta}^{(\ell)}} = \tilde{h}_{\{W^{(\ell)},\boldsymbol{b}^{(\ell)}\}}. \tag{2}$$

We will focus on the fixed-model capacity neural network setup where the architectural parameters L and $\{n_\ell\}_{\ell\in[L]}$, along with the activation function ϕ are fixed before training. Some popular choices of activation function include the hyperbolic tangent, the sigmoid function, a rectified linear unit, and a linear map. The choice of weight matrices $W^{(\ell)}$ and bias vectors $\boldsymbol{b}^{(\ell)}$ proceeds through optimization-based training; we seek to choose $\tilde{\theta}$ to minimize an ℓ^2-type loss function that balances model complexity $R(\tilde{\theta})$ against fidelity to available training data $(\boldsymbol{x}_m, \boldsymbol{y}_m)_{m\in[M]}$,

$$\mathcal{L}\left(\tilde{\theta}\right) = \sum_{m=1}^{M} \frac{1}{2}\mathcal{L}_m\left(\tilde{\theta}\right) + \frac{\lambda}{2}R\left(\tilde{\theta}\right) = \sum_{m=1}^{M} \frac{1}{2}\left\|\boldsymbol{y}(\boldsymbol{x}_m) - \boldsymbol{y}_m\right\|_2^2 + \frac{\lambda}{2}R\left(\tilde{\theta}\right)$$

where $\lambda > 0$ is a tunable hyperparameter. In this paper, we choose R as a Tikhonov-type regularization,

$$R\left(\tilde{\theta}\right) = \sum_{\ell=2}^{L}\left(\left\|W^{(\ell)}\right\|_F^2 + \left\|\boldsymbol{b}^{(\ell)}\right\|_2^2\right). \tag{3}$$

Minimization of $\mathcal{L}$ over the optimization variables $\tilde{\theta}$ proceeds typically with first-order or quasi-Newton methods, so that computation of $\frac{\partial \mathcal{L}}{\partial \tilde{\theta}}$ is required. Practical algorithms achieve this through back-propagation, summarized by the iteration,

$$W^{(L+1)T}\boldsymbol{\delta}_m^{(L+1)} \coloneqq \boldsymbol{y}(\boldsymbol{x}_m) - \boldsymbol{y}_m, \qquad \boldsymbol{\delta}^{(\ell)} = \phi'(W^{(\ell)}\boldsymbol{a}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}) \circ W^{(\ell+1)T}\boldsymbol{\delta}^{(\ell+1)},$$

for $\ell = L, \ldots, 2$, where $\circ$ between vectors denotes a componentwise (Hadamard) product, $\phi' : \mathbb{R} \to \mathbb{R}$ is the derivative of ϕ, and application to vectors is again defined componentwise. This results in the relations,

$$\frac{\partial \mathcal{L}_m}{\partial W^{(\ell)}} = \boldsymbol{\delta}^{(\ell)}\boldsymbol{a}^{(\ell)T}, \qquad \frac{\partial \mathcal{L}_m}{\partial \boldsymbol{b}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)}.$$

Gradient descent, applied with learning rate η, proceeds via the update,

$$W^{(\ell)} \leftarrow W^{(\ell)} - \eta \sum_{m=1}^{M} \frac{\partial \mathcal{L}_m}{\partial W^{(\ell)}} - \eta\lambda W^{(\ell)}, \qquad \boldsymbol{b}^{(\ell)} \leftarrow \boldsymbol{b}^{(\ell)} - \eta \sum_{m=1}^{M} \frac{\partial \mathcal{L}_m}{\partial \boldsymbol{b}^{(\ell)}} - \eta\lambda b^{(\ell)}.$$

In all the expressions above, application of matrix-vector multiplications involving $W^{(\ell)}$ can form a substantial portion of the computational burden, especially if the hidden layers have large dimension n_ℓ. In this manuscript, we seek to alleviate this burden while retaining model capacity.

3. The Kronecker Product As matrix and vector sizes increase, matrix-vector operations require more computational resources and time; the Kronecker product [18] is one strategy to ameliorate this complexity when the matrices involved have a certain type of exploitable structure. Given $L \in \mathbb{R}^{m_1 \times n_1}$ and $R \in \mathbb{R}^{m_2 \times n_2}$, the Kronecker product $L \otimes R$ is defined as,

$$K \coloneqq L \otimes R = \begin{bmatrix} l_{11}R & \cdots & l_{1n_1}R \\ \vdots & \ddots & \vdots \\ l_{m_1 1}R & \cdots & l_{m_1 n_1}R \end{bmatrix} \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}. \tag{4}$$

Given $\boldsymbol{x} \in \mathbb{R}^{n_1 n_2}$, computing $K\boldsymbol{x}$ can be accomplished via the relation,

$$K\boldsymbol{x} = RXL^T, \qquad X = \mathrm{mat}(\boldsymbol{x}), \tag{5}$$

where $\mathrm{mat} : \mathbb{R}^{n_1 n_2} \to \mathbb{R}^{n_2 \times n_1}$ is a matricization operation, the inverse of vectorization $\mathrm{vec} : \mathbb{R}^{n_2 \times n_1} \to \mathbb{R}^{n_1 n_2}$. The major appeal of the above representation is that $\boldsymbol{x} \mapsto K\boldsymbol{x}$ requires $\mathcal{O}(m_1 m_2 n_1 n_2)$ operations, whereas $X \mapsto RXL^T$ requires only $\mathcal{O}(n_2 n_1 m_2 + n_1 n_2 m_1)$ operations, which can result in substantial computational acceleration.

While many matrices cannot be represented exactly as a Kronecker product, all matrices whose row and column dimensions are not prime integers can be approximated by a sum of Kronecker product matrices. To explain this further, let $W \in \mathbb{R}^{m \times n}$ be given, with $m = m_1 m_2$ and $n = n_1 n_2$ arbitrary integer factorizations of m and n. A rank-k KPD approximation of W is,

$$W \approx \sum_{j=1}^{k} L^{(j)} \otimes R^{(j)}, \qquad L^{(j)} \in \mathbb{R}^{m_1 \times n_1},\ R^{(j)} \in \mathbb{R}^{m_2 \times n_2},$$

for some choice of matrices $L^{(j)}, R^{(j)}$. Note that the phrase *rank-k KPD approximation* is a slight abuse of terminology, as $L^{(j)} \otimes R^{(j)}$ is not necessarily a rank-1 matrix. A Frobenius-norm optimal rank-k KPD approximation can be determined by introducing a *rearrangement* $\mathcal{R}$ of W, identified through a block partition of W,

$$W = \begin{bmatrix} W_1 & \cdots & W_{m_1(n_1-1)+1} \\ \vdots & \ddots & \vdots \\ W_{m_1} & \cdots & W_{m_1 n_1} \end{bmatrix}, \; W_j \in \mathbb{R}^{m_2 \times n_2} \quad \overset{\boldsymbol{w}_j = \mathrm{vec}(W_j)}{\Longrightarrow} \quad \mathcal{R}(W) = \begin{bmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_{m_1 n_1}^T \end{bmatrix}.$$

With $\boldsymbol{l} = \mathrm{vec}(L)$ and $\boldsymbol{r} = \mathrm{vec}(R)$, then note that $\mathcal{R}(L \otimes R) = \boldsymbol{l}\boldsymbol{r}^T$, and therefore $\|W - L \otimes R\|_F = \left\|\mathcal{R}(W) - \boldsymbol{l}\boldsymbol{r}^T\right\|_F$. This last relation allows one to leverage the Hilbert-Schmidt-Eckart-Young theorem on Frobenius-norm optimal low-rank approximations using the singular value decomposition [6, Theorem 2.4.8]. While this immediately yields an optimal rank-1 KPD approximation [18, Corrolary 2.2], the result is generalizable to rank-k approximations for $k > 1$.

Lemma 3.1 ([17]). *Given $W \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}$, let its rearrangement have singular value decomposition (SVD) $\mathbb{R}^{m_1 n_1 \times m_2 n_2} \ni \mathcal{R}(W) = U\Sigma V^T = \sum_{i=1}^r \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$, with the singular values $\{\sigma_i\}_{i\in[r]}$ arranged in non-increasing order and $r = \mathrm{rank}(\mathcal{R}(W))$. Let W_k denote a rank-k KPD defined by,*

$$W_k \coloneqq \sum_{j=1}^k \mathrm{mat}(\sigma_i \boldsymbol{u}_i) \otimes \mathrm{mat}(\boldsymbol{v}_i).$$

Then W_k is an optimal rank-k KPD approximation to W:

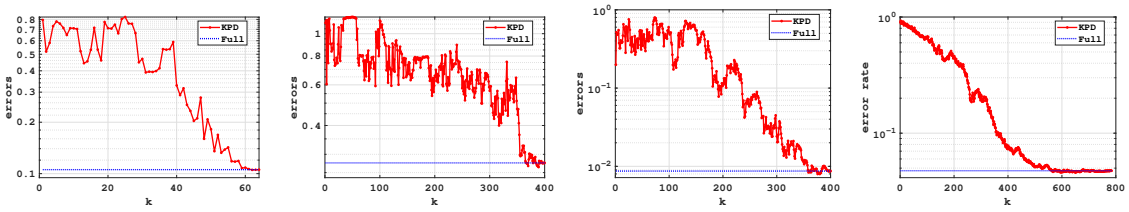
$$W_k \in \underset{L^{(j)} \in \mathbb{R}^{m_1 \times n_1}, R^{(j)} \in \mathbb{R}^{m_2 \times n_2}, j \in [k]}{\arg\min} \left\| W - \sum_{j=1}^k L^{(j)} \otimes R^{(j)} \right\|_F^2,$$

$$\|W - W_k\|_F^2 = \sum_{i=k+1}^r \sigma_i^2.$$

4. KP Dual Layer Networks This section introduces our new neural network architecture. Given an FNN with layer ℓ nodal states $\boldsymbol{a}^{(\ell)}$ defined by (1), a KPD $W^{(\ell)} = \sum_{i=1}^r L^{(\ell,i)T} \otimes R^{(\ell,i)}$, with $r = \mathrm{rank}(\mathcal{R}(W^{(\ell)}))$ could be used to produce a new intermediate layer utilizing KP-based multiplication. I.e., we have the exact representation,

$$\boldsymbol{a}^{(\ell)} = \tilde{h}_{\tilde{\theta}^{(\ell)}}(\boldsymbol{a}^{(\ell-1)}) = \mathrm{vec}\left(\sum_{i=1}^r R^{(\ell,i)} A^{(\ell-1)} L^{(\ell,i)} \right) + \boldsymbol{b}^{(\ell)},$$

where $A^{(\ell-1)} = \mathrm{mat}\left(\boldsymbol{a}^{(\ell-1)}\right)$ is reshaped to form a matrix with appropriate dimensions. A straightforward idea is then to truncate the sum to $k < r$ terms. However, the error incurred by such an approach is bounded by the truncated singular values of the weight rearrangement, and such singular values are not guaranteed to decay quickly, in turn requiring large k to accurately represent $W^{(\ell)}$. Figure 1 demonstrates this for examples that will be introduced in Table 3. For each of these examples, a rank close to the full rank of the rearrangement of the weight matrices is needed to achieve a test error similar to the test error when using the trained weight matrices.



(a) BSD (a) (b) BSD (b) (c) BF (d) MNIST

Figure 1: Figures 1a to 1d show the test errors across choice of KPD rank, k, compared to the full rank of the rearrangement of each weight matrix for BSD (a and b), BF, and MNIST respectively as defined in Table 3.

To balance this potential loss of accuracy caused by reducing the Kronecker rank, we augment model capacity via inclusion of an extra activation function and bias vector at the new intermediate layer. With $A_R^{(\ell)}$ the new matrix-valued intermediate state, and with ϕ_1 and ϕ_2 activation functions, then we define a rank-k KP dual layer (KDL) as the operation,

$$A_R^{(\ell)} = \left(\phi_1 \circ h_{\theta_R^{(\ell)}} \circ \phi_2 \circ \widetilde{h}_{\theta_L^{(\ell)}}\right)\left(A_R^{(\ell-1)}\right),$$

where we have introduced new layer-ℓ parameters $\theta_L^{(\ell)}$ and $\theta_R^{(\ell)}$,

$$\begin{aligned}
\theta_L^{(\ell)} &:= \left\{W_L^{(\ell,1)}, B_L^{(\ell,1)}, \cdots, W_L^{(\ell,k)}, B_L^{(\ell,k)}\right\}, \\
\theta_R^{(\ell)} &:= \left\{W_R^{(\ell,1)}, B_R^{(\ell,1)}, \cdots, W_R^{(\ell,k)}, B_R^{(\ell,k)}\right\}, \\
\theta &:= \left\{\theta_L^{(2)}, \theta_R^{(2)}, \cdots, \theta_L^{(L)}, \theta_R^{(L)}\right\}.
\end{aligned} \tag{6}$$

The function $\tilde{h}_{\theta_L^{(\ell)}}$ is an extension of the affine function introduced in (2)*, and the newly introduced functions h_θ, with a slight abuse in notation, are k-fold sums / collections of FNN affine maps $\widetilde{h}$,

$$\begin{aligned}
\widetilde{h}_{\theta_L^{(\ell,i)}}\left(A_R^{(\ell-1)}\right) &= A_R^{(\ell-1)} W_L^{(\ell,i)} + B_L^{(\ell,i)}, \\
h_{\theta_L^{(\ell)}}\left(A_R^{(\ell-1)}\right) &= \left(\widetilde{h}_{\theta_L^{(\ell,1)}}\left(A_R^{(\ell-1)}\right), \ldots, \widetilde{h}_{\theta_L^{(\ell,k)}}\left(A_R^{(\ell-1)}\right)\right) \\
A_L^{(\ell,i)} &= \left(\phi_2 \circ \widetilde{h}_{\theta_L^{(\ell,i)}}\right)\left(A_R^{(\ell-1)}\right),
\end{aligned} \tag{7}$$

$$\begin{aligned}
h_{\theta_R^{(\ell)}}\left(\left\{A_L^{(\ell,i)}\right\}_{i=1}^k\right) &= \sum_{i=1}^k \widetilde{h}_{\theta_R^{(\ell,i)}}\left(A_L^{(\ell,i)}\right) = \sum_{i=1}^k \left(W_R^{(\ell,i)} A_L^{(\ell,i)} + B_R^{(\ell,i)}\right), \\
A_R^{(\ell)} &= \left(\phi_1 \circ h_{\theta_R^{(\ell)}} \circ \phi_2 \circ h_{\theta_L^{(\ell)}}\right)\left(A_R^{(\ell-1)}\right),
\end{aligned} \tag{8}$$

*$\widetilde{h}$ in (2) takes a single weight matrix and bias vector as parameters, but $\widetilde{h}$ here takes k weight matrices and bias vectors as parameters.

effectively splitting the layer into two intermediate states $A_L^{(\ell,i)}$, $i \in [k]$, and $A_R^{(\ell)}$.

In what follows, we will use the notation,

$$f_\ell(A) = \left(\phi_1 \circ h_{\theta_R^{(\ell)}} \circ \phi_2 \circ \tilde{h}_{\theta_L^{(\ell)}}\right)(A), \tag{9a}$$

to denote the function that transitions the nodes at layer $\ell - 1$ to those at layer ℓ for $\ell \geq 2$. Note that reshaping between layers is only necessary if the output dimensions from one layer do not match the input dimensions from the next layer. For notational convenience in our analysis, we define,

$$f_1(\cdot) = X, \tag{9b}$$

and in particular we have $f_1(0) = X$. A higher order splitting into Kronecker multi-layers (KML) is discussed in supplementary materials, and is hierarchical in nature. While KDLs may be applied only to a subset of layers in an FNN, of interest here is a neural network based solely on KDLs, i.e., a KDL-NN, where intermediate layers are left in matrix form (instead of vectorizing), with input $\mathrm{mat}(\boldsymbol{x})$ denoted by X. The KDL-NN is now given by

$$Y_k^\kappa(X) = Y_k^\kappa(X;\theta) = (f_L \circ f_{L-1} \cdots \circ f_2)(X). \tag{10}$$

The associated regularized loss for training a rank-k KDL-NN over all data points $(\boldsymbol{x}_m, \boldsymbol{y}_m)$ for $m = 1, \ldots, M$, is given by,

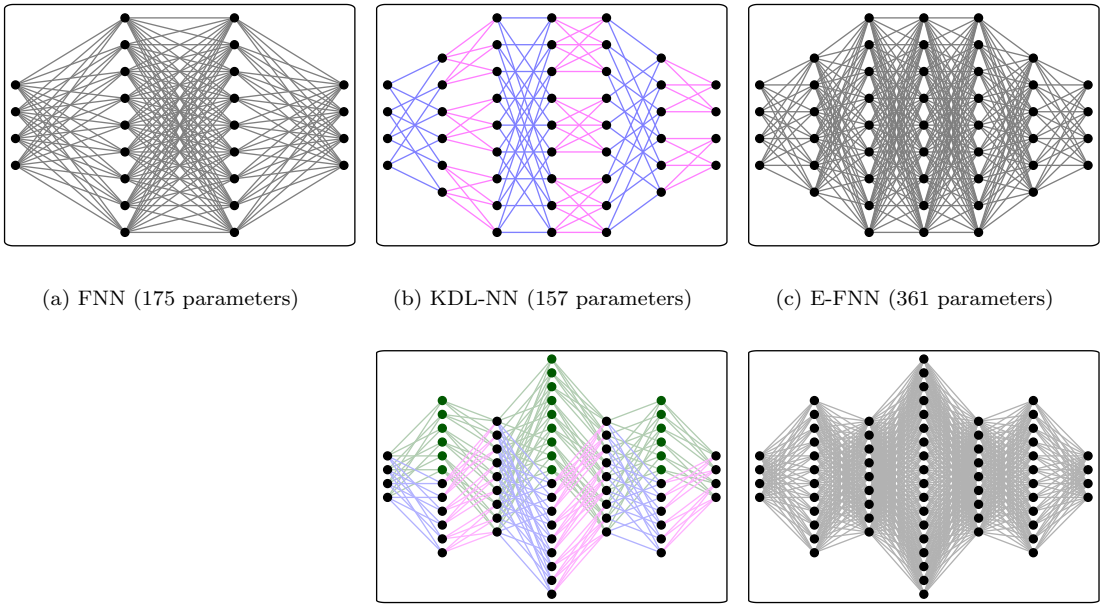
$$\mathcal{L}(\theta) = \sum_{m=1}^{M} \frac{1}{2}\mathcal{L}_m^\kappa(\theta) + \frac{\lambda}{2}R^\kappa(\theta) = \sum_{m=1}^{M} \|\mathrm{mat}(\boldsymbol{y}_m) - Y_k^\kappa(\mathrm{mat}(\boldsymbol{x}_m))\|_F^2 + \frac{\lambda}{2}R^\kappa(\theta),$$

where R^κ is the same Tikhonov-type regularizer as in (3), but sums the squared Frobenius norms for all weights and bias matrices contained in θ.

4.1. KDL-NN architecture In what follows, we use notation that describes node configuration as a sequence of numbers that count nodes in each layer, e.g., the FNN in Figure 2a corresponds to $4|9|9|4$, describing a fully connected network with 4 input nodes, 2 hidden layers with 9 nodes each, and an output layer with 4 nodes. A rank-1 KDL-NN may be described by using pairs of values for each layer corresponding to the matricized shape of the nodes, such as $(2,2)|(3,3)|(3,3)|(2,2)$, where the input and output layers with 4 nodes are shaped as 2×2 matrices and the two hidden layers with 9 nodes each are shaped as 3×3 matrices; the layers added by the KDL-NN compared to the FNN architecture correspond to matrices of size 2×3, 3×3 and 3×2, respectively. We denote a rank-k KDL-NN with similar notation, $(2,2)|^k(3,3)|^k(3,3)|^k(2,2)$, indicating that k factors of each of the 2×3, 3×3, and 3×2 added intermediate layers are present. (A bar $|$ without a superscript indicates $k = 1$.) In our numerical examples, we provide comparison against a third type of network, an "extended" FNN, E-FNN, corresponding to the same number of nodes as the KDL-NN, but with full connections between nodes in sequential layers. An E-FNN corresponding to the same number of nodes as the KPD-NN architecture $(2,2)|(3,3)|(3,3)|(2,2)$ then corresponds to $4|6|9|9|9|6|4$.

We visually compare the architecture of the above FNN, KDL-NN, and E-FNN examples in Figure 2 for ranks $k = 1$ and $k = 2$. The KDL-NN sparsifies some connections of the FNN while adding more nodes. Alternatively, the KDL-NN is a connection sparsification of the E-FNN.

Note that we provide comparisons against E-FNN's as a baseline for model capacity; our goal is not to computationally sparsify E-FNN's, but rather to build new architectures based on simpler FNN's. Table 1 summarizes architectures described above and reports the total number of trainable parmeters for the networks shown in Figure 2. Note that KDL-NN's are fully connected across each set of dual layers.



(a) FNN (175 parameters) (b) KDL-NN (157 parameters) (c) E-FNN (361 parameters)

(d) KDL-NN, $k = 2$ (292 parameters) (e) E-FNN, $k = 2$ (700 parameters)

Figure 2: Top row: A fully connected FNN is shown in Figure 2a, and Figure 2b shows the resulting connections for reshaping the input and output into 2×2 blocks and the hidden layers into 3×3 blocks with intermediate layers added corresponding to a KDL-NN rank 1 formulation , where the alternating pattern represents multiplying from the right shown with blue connections and from the left with maroon connections. The same nodes from the KDL-NN are then used to show the resulting E-FNN in Figure 2c. The bottom row corresponding to a KPD-NN rank 2 formulation (Figure 2d), and are plotted with the additional nodes and connections plotted in green. The same nodes are then used to show the resulting E-FNN in Figure 2e. The architecture and total number of parameters for all networks pictured here are given in Table 1.

4.2. Truncation error analysis for KDL-NN architectures In this section, we analyze theoretical performance of *simplified* KDL-NN architectures based on the Kronecker product approximation error estimate in Lemma 3.1. Because we use these estimates that are based on a truncated SVD, we expect the performance of a trained KDL-NN to perform much better. In addition, the simplified networks have smaller model capacity than the practical networks we use (as described next). Nevertheless, these results show that KDL-NN's can inherit accuracy from the approxima-

Network type	Architecture	Number of parameters
FNN	4|9|9|4	175
KDL-NN	$(2,2)\|(3,3)\|(3,3)\|(2,2)$	157
corresponding E-FNN	4|6|9|9|9|6|4	361
KDL-NN ($k=2$)	$(2,2)\|^2(3,3)\|^2(3,3)\|^2(2,2)$	292
corresponding E-FNN	4|12|9|18|9|12|4	700

Table 1: Architectures and parameter counts for networks shown in Figure 2.

bility of fully connected weight matrices by Kronecker sums.

We recall that our KDL-NN architectures involve two activation functions ϕ_1 and ϕ_2 in (8). The *simplified* networks we analyze below take ϕ_2 as the identity (linear) function, which allows us to directly tie approximation error to Kronecker sum truncation errors of matrices. In practice (i.e., in our numerical results) we use non-identity activation functions.

To state our results we require a norm on the KDL-NN parameters (6). Consider $\theta^{(\ell)}$, the KDL-NN parameters associated to layer ℓ,

$$\theta^{(\ell)} = \left\{ \theta_L^{(\ell)}, \theta_R^{(\ell)} \right\},$$

and introduce a particular function $|\cdot|_\kappa$ operating on such layer-ℓ parameters,

$$\left|\theta^{(\ell)}\right|_\kappa^2 := \sum_{q=1}^{k} \left\| W_L^{(\ell,q)} \right\|_F \left\| W_R^{(\ell,q)} \right\|_F.$$

It is straightforward to show that this function is non-negative but dominated by the function $\theta^{(\ell)} \mapsto \left\|\mathrm{vec}(\theta^{(\ell)})\right\|_2$, so that $|\cdot|_\kappa$ is weaker than a corresponding standard ℓ^2 norm on the vectorized parameters.

We assume that an FNN architecture is given, and that an associated KDL-NN architecture is prescribed (e.g., so that we have a well-defined matricization of input/output vectors). Our results are relative to a *trained* FNN $\boldsymbol{y}$ in (2), which we express in matricized form:

$$Y = Y(X), \qquad Y = \mathrm{mat}(\boldsymbol{y}), \qquad X = \mathrm{mat}(\boldsymbol{x}).$$

This trained FNN has weight matrices $W^{(\ell)}$. By constructing a KDL-NN whose parameters correspond to approximating $W^{(\ell)}$ via Kronecker product sums, one expects that the classical Kronecker sum bounds in Lemma 3.1 can be leveraged for error estimates in the KDL-NN case. Our results appear in terms of ℓ^2 norms of truncated singular values from a Kronecker product rearrangement:

$$\epsilon^{(\ell,k)} := \sqrt{\sum_{i=k+1}^{r^{(\ell)}} (\sigma_i^{(\ell)})^2} \overset{\text{Lemma 3.1}}{=} \min_{L^{(j)} \in \mathbb{R}^{m_1 \times n_1}, R^{(j)} \in \mathbb{R}^{m_2 \times n_2}, j \in [k]} \left\| W^{(\ell)} - \sum_{j=1}^{k} L^{(\ell,j)} \otimes R^{(\ell,j)} \right\|_F^2 \tag{11}$$

where $\left\{\sigma_i^{(\ell)}\right\}_{i \in r^{(\ell)}}$ are the ordered singular values of the rank-$r^{(\ell)}$ rearrangement of FNN weight matrix $W^{(\ell)}$. Our main technical result characterizes errors for a simplified KDL-NN relative to an FNN using the "norms" $\left|\theta^{(\ell)}\right|_\kappa$ and the Kronecker rank truncation parameters $\epsilon^{(\ell)}$.

Theorem 4.1. *Suppose a trained FNN Y is given, and consider a rank-k KDL-NN network Y_k^κ in (10) where we choose the inner activation function $\phi_2(x) \equiv x$. Then given a data pair $(\boldsymbol{x}_m, \boldsymbol{y}_m)$ with corresponding matricization (X_m, Y_m), training Y_k^κ over the KDL-NN parameters θ yields,*

$$\arg\min_\theta \|Y(X) - Y_k^\kappa(X;\theta)\|_F \leq \sum_{i=2}^{L} \epsilon^{(i,k)} \left(\prod_{j=i+1}^{L} \left|\theta^{(j)}\right|_\kappa^2 \right) \sum_{k=1}^{i-1} c_1^{L-k} \|f_k(0)\|_F . \tag{12}$$

where the parameters $\theta^{(\ell)}$ are defined by setting $W_L^{(\ell,i)} = L^{(\ell,i)}$ and $W_R^{(\ell,i)} = R^{(\ell,i)}$, with $L^{(\ell,i)}$ and $R^{(\ell,i)}$ defined as in (11) and f_1 is as defined in (9b).

Under some additional assumptions, the above can be simplified to more clearly reveal the components of the bound.

Corollary 4.1.1. *Suppose $f_\ell(0) = 0$ for $\ell = 2, \ldots, L$ and the activation function $\phi_1(\boldsymbol{x}) = \phi(x)$ is 1-Lipschitz (ReLu, Tanh, and Sigmoid are such examples [15]), then Theorem 4.1 holds with*

$$\arg\min_\theta \|Y(X) - Y_k^\kappa(X;\theta)\|_F \leq \left(\sum_{i=2}^{L} \epsilon^{(i,k)} \prod_{j=i+1}^{L} \left|\theta^{(j)}\right|_\kappa^2 \right) \|X\|_F .$$

We provide the proof of Theorem 4.1 in Appendix B. This theorem provides a theoretical connection between the size of the KPD truncation errors $\epsilon^{(i,k)}$ of a trained FNN and predictive performance of a KDL-NN relative to this FNN. This result does *not* immediately translate into a practical error estimate since (a) we have made the simplifying assumption that ϕ_2 is the identity, and (b) training a KDL-NN does *not* involve KPD truncations from weights of an FNN. We also do not expect this bound to be sharp since its proof (see Appendix B) invokes the triangle inequality several times.

Nevertheless, the components of the estimate in (12) give insight into when we expect KDL-NN approaches to work well: First, if all the $\epsilon^{(i,k)}$, $i = 2, \ldots, L$ are small, then we expect that a KDL-NN can perform at least as well as a corresponding FNN. I.e., when trained weight matrices of an FNN have "small" Kronecker rank, we expect KDL-NN's to perform well. The remaining terms can be interpreted as quantities that measure how well-behaved a KDL-NN is. For example, appearance of the $|\theta^{(j)}|_\kappa$ functions indicates that the size of the weight matrices affects performance, and $f_k(0)$ is the output of a layer-k KDL function with zero input. Note in particular that $f_1(0) = X$, so that the norm of the input X to the KDL-NN affects the bound, as expected.

4.3. Numerical Cost of Forward Operations and Back-Propagation We now discuss the computational cost of a KDL-NN and give a broad technical explanation of why we expect the KDL-NN to be more efficient in practice. Given a KDL-NN defined by (10), gradient descent updates are performed on layer ℓ from L to 2 via the relations,

$$\begin{aligned}
W_R^{(\ell,i)} &\leftarrow (1 - \lambda\eta) W_R^{(\ell,i)} - \eta \Delta_1^{(\ell,i)} A_L^{(\ell,i)T} \\
W_L^{(\ell,i)} &\leftarrow (1 - \lambda\eta) W_L^{(\ell,i)} - \eta A_R^{(\ell-1)T} \Delta_2^{(\ell,i)} \\
B_R^{(\ell,i)} &\leftarrow (1 - \lambda\eta) B_R^{(\ell,i)} - \eta \Delta_1^{(\ell,i)} \\
B_L^{(\ell,i)} &\leftarrow (1 - \lambda\eta) B_L^{(\ell,i)} - \eta \Delta_2^{(\ell,i)},
\end{aligned}$$

where the intermediate matrices $\Delta_1^{(\ell,i)}$, $\Delta_2^{(\ell,i)}$ and $\Gamma^{(\ell)}$ are defined in Appendix A. A simple implementation of back-propagation with learning rate η for KDL pair ℓ from layer pairs L to 2 is given in Algorithm 1.

Initialize $\Gamma^{(\ell-1)} = 0$
for $i = 1$ to k **do**
  $\Delta_1^{(\ell,i)} = \phi_1'(Z_R^{(\ell,i)}) \circ \Gamma^{(\ell)}$
  $\Delta_2 = ((W_R^{(\ell,i)})^T \Delta_1^{(\ell,i)}) \circ \phi_2'(Z_L^{(\ell,i)})$
  $\Gamma^{(\ell-1)} \leftarrow \Gamma^{(\ell-1)} + \Delta_2^{(\ell,i)} (W_L^{(\ell,i)})^T$
  $W_R^{(\ell,i)} \leftarrow (1-\lambda) W_R^{(\ell,i)} - \eta \Delta_1^{(\ell,i)} (A_L^{(\ell,i)})^T$
  $W_L^{(\ell,i)} \leftarrow (1-\lambda) W_L^{(\ell,i)} - \eta (A_R^{(\ell-1)T} \Delta_2^{(\ell,i)})$
  $B_R^{(\ell,i)} \leftarrow (1-\lambda) B_R^{(\ell,i)} - \eta \Delta_1^{(\ell,i)}$
  $B_L^{(\ell,i)} \leftarrow (1-\lambda) B_L^{(\ell,i)} - \eta \Delta_2^{(\ell,i)}$
end for

Algorithm 1: KDL Back-Propagation

The cost of forward operations for an FNN layer (1) with $W \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}$ is dominated by $\mathcal{O}(m_1 m_2 n_1 n_2)$ flops and the cost of the activation function operating on $m_1 m_2$ values. In addition, back-propagation requires action by ϕ' on $m_1 m_2$ values, and is then dominated by $\mathcal{O}(m_1 m_2 n_1 n_2)$ flops for the update.

In comparison, the KDL-NN formulation in (10) is dominated by $\mathcal{O}(m_2 n_1 (n_2 + m_1))$ in total, with activation function operating on $m_1 n_2$ and $m_1 m_2$ elements respectively. Back-propagation then requires action by ϕ' on $m_1 n_2$ and $m_1 m_2$ values respectively. Updates on the KDL weight matrices are dominated by $\mathcal{O}(m_1 n_2 (m_2 + n_1))$ flops. The dominant cost for the KDL-NN's is minimized when $m_2 \approx n_1 \approx m_1 \approx n_2$, reflecting the same savings that one achieves in matrix multiplication involving reshaping of Kronecker product matrices. We show in our results that these savings are considerable in practice.

5. Numerical Results We compare our deep learning performance (training cost and test data accuracy) for our novel KDL-NN architecture against FNN's and E-FNN's. The networks are trained first on the function

$$f(\boldsymbol{x}) = \left(\frac{\prod_{k=1}^{\lceil n_1/2 \rceil} \left(1 + 4^k x_k^2\right)}{\prod_{k=\lceil n_1/2 \rceil + 1}^{n_1} \left(100 + 5 x_k\right)} \right)^{\frac{1}{d}}$$

evaluted for normal random $x_k \in (-1, 1)$ for $k = 1, \cdots, n_1$, similar to [1, 4], and then on the Bike Sharing Dataset (BSD), [5], the BlogFeedback data set (BF), [3], and the MNIST data set, [12]. We summarize the sizes of these data sets and the number of inputs/outputs in Table 2.

While M data points are used to train the networks, $\widetilde{M}$ points $(\widetilde{\boldsymbol{x}}_m, \widetilde{\boldsymbol{y}}_m)_{m \in [\widetilde{M}]}$ used as test data to determine performance on unseen inputs. We report standard ℓ^2 test losses,

$$\mathcal{L} = \frac{1}{\widetilde{M}} \sum_{m=1}^{\widetilde{M}} \|\widetilde{y}_m - \widetilde{y}_*(\widetilde{x}_m)\|_2^2 ,$$

Data set	Inputs n_1	Outputs n_L	$(M, \widetilde{M})$
$f(\boldsymbol{x})$ [1, 4]	8	1	(10000, 1000)
BSD [5]	14	1	(13903, 3476)
BF [3]	280	1	(41918, 10479)
MNIST [12]	784	10	(60000, 10000)

Table 2: Available data and input/output sizes n_1/n_L for the examples in this paper. Also shown are the number of training/test points $M/\widetilde{M}$.

where $\widetilde{y}_*$ is the (vectorized) output of an FNN, E-FNN, or KDL-NN architecture. All tests are run on a system with a 2.10GHz $\times 64$ processor with 125.5 GiB memory using MATLAB R2021a.

In all examples, we prescribe an FNN architecture, make choices for integer factorizations of input and hidden layer sizes (e.g., $9 = 3 \times 3$), and derive a corresponding KDL-NN and subsequently E-FNN architecture. The discussion in subsection 4.3 motivates that the choice of integer factorization should maximize the geometric mean of the factors. The ReLu activation function is used for training $f(\boldsymbol{x})$, and tanh is used for all other examples.

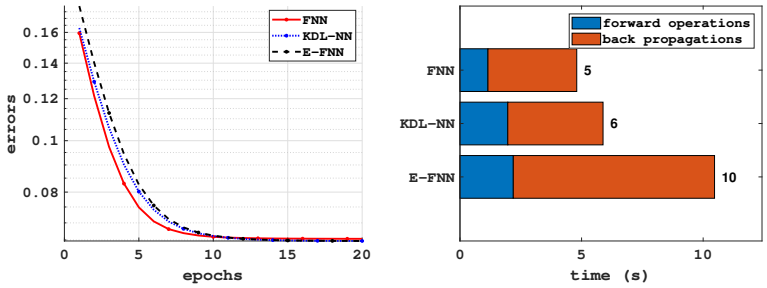
5.1. Fixed-rank KDL-NN performance We demonstrate the efficacy of KDL-NN architectures with fixed Kronecker rank k. Errors and timing are shown for $f(\boldsymbol{x})$ with rank 1 KDL-NN in Figure 3, and for BSD, BF, and MNIST in Figure 4 with ranks 1 and 2 KDL-NN's. All data is summarized in Table 3.

For matrix-vector multiplication, the Kronecker product operations (5) significantly improve practical efficiency when the matrices L and R are large, but not when they are small. This property extends to our KDL-NN architecture. We demonstrate this by prescribing two different FNN architectures for the BSD dataset: BSD(a) corresponds to an FNN where the L and R matrices have "small" sizes, and BSD(b) to one where they have large sizes. The results in Figure 4 show that for the BSD(a) architecture, the FNN is more efficient to train than the KDL-NN network due to the small sizes of the factorized matrices. However, for BSD(b), we increase the network size and observe that KDL-NN training is much faster. These observations are consistent with what one would expect for Kronecker product-based matrix multiplication.

In terms of accuracy, we see that the KDL-NN architecture tends to maintain or improve model capacity compared to FNN architectures, even for rank 1 KDL-NN's.

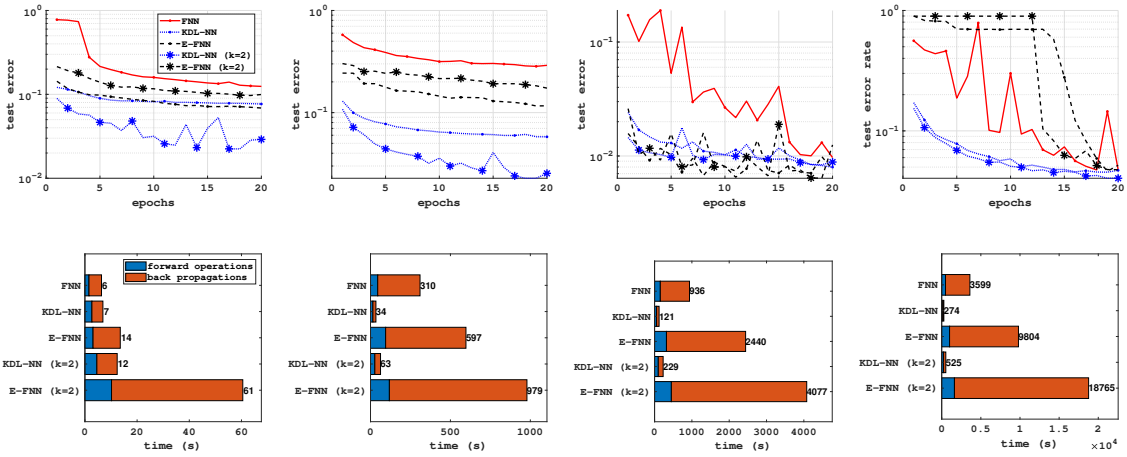
5.2. Adaptive Choice of Rank In general, the rank needed to optimize the KDL-NN is unknown. Since the KDL summands use separate weight matrices, it is straightforward to add new pairs of weight matrices during training to increase the rank.

To check the decay of the errors, a validation set is pulled from the training set based on 10% of the total set size. New pairs of weight matrices are initialized to normal random matrices, scaled to machine epsilon, and added when the decay of a range in the validation error levels off. In order to achieve a reduction in error when adding matrices, the learning rate may need to be adjusted. Factors are chosen in a range from $\frac{1}{n}$ to 2. A learning rate based on each factor is then used for a set number of epochs, and the learning rate that produces the smallest error is selected moving forward. Figure 5 shows training errors and timing using this procedure with a range of 4 learning



(a) $f(\boldsymbol{x})$ (b) $f(\boldsymbol{x})$

Figure 3: Figures 3a and 3b show the test errors and timing, broken down by forward operations and back-propagations, for $f(\boldsymbol{x})$ using networks defined in Table 3 with Kronecker rank 1.



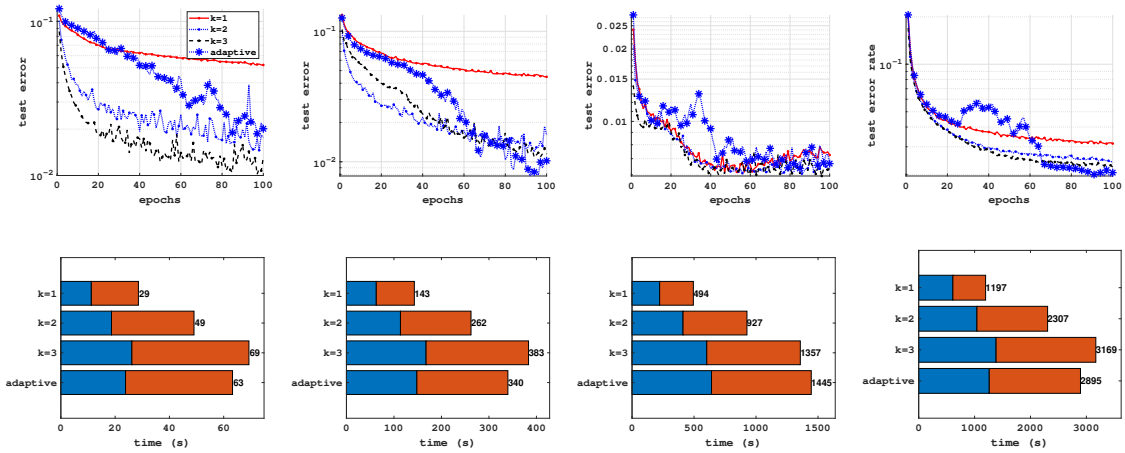
(a) BSD (a) (b) BSD (b) (c) BF (d) MNIST

Figure 4: Figures 4a to 4d show the test errors and timing, broken down by forward operations and back-propagations, for FNN, KDL-NN, and E-FNN for BSD (a and b), BF, and MNIST respectively as defined in Table 3 with Kronecker ranks $k = 1$ and $k = 2$.

Data set	Network	Architecture	# Parameters	→	←	Total time	Test error (%)
	FNN	8|64|64|1	4,801	**1.2**	**3.7**	**4.8**	6.53
$f(\boldsymbol{x})$	KDL-NN	$(2,4)\|(8,8)\|(8,8)\|(1,1)$	**409**	2.0	3.9	5.9	6.47
	E-FNN	8|16|64|64|64|8|1	10,081	2.2	8.3	10.5	**6.47**
	FNN	14|64|64|1	5,185	**1.6**	4.8	**6.4**	12.45
	KDL-NN	$(2,7)\|(8,8)\|(8,8)\|(1,1)$	**433**	2.7	**4.2**	6.9	7.68
BSD (a)	E-FNN	14|16|64|64|64|8|1	10,177	3.1	10.4	13.6	6.87
	KDL-NN	$(2,7)\|^2(8,8)\|^2(8,8)\|^2(1,1)$	866	4.6	7.8	12.4	**2.91**
	E-FNN	14|32|64|128|64|16|1	20,225	10.2	50.3	60.5	9.96
	FNN	14|400|400|1	166,801	45.1	264.7	309.8	29.03
	KDL-NN	$(2,7)\|(20,20)\|(20,20)\|(1,1)$	**2,281**	**14.9**	**19.0**	**33.9**	5.79
BSD (b)	E-FNN	14|40|400|400|400|20|1	345,841	94.6	502.6	597.2	11.67
	KDL-NN	$(2,7)\|^2(20,20)\|^2(20,20)\|^2(1,1)$	4,562	27.1	35.6	62.7	**2.54**
	E-FNN	14|80|400|800|400|40|1	690,881	118.1	860.7	978.8	17.36
	FNN	280|400|400|1	273,201	153.7	782.3	935.9	0.97
	KDL-NN	$(20,14)\|(20,20)\|(20,20)\|(1,1)$	**3,141**	**54.1**	**66.9**	**121.0**	0.80
BF	E-FNN	280|400|400|400|400|20|1	601,641	318.6	2,121.1	2,439.7	**0.76**
	KDL-NN	$(20,14)\|^2(20,20)\|^2(20,20)\|^2(1,1)$	6,282	99.7	129.8	229.4	0.88
	E-FNN	280|800|400|800|400|40|1	1,202,481	449.8	3,627.6	4,077.4	1.25
	FNN	784|784|784|10	1,238,730	486.5	3,112.8	3,599.4	4.75
	KDL-NN	$(28,28)\|(28,28)\|(28,28)\|(5,2)$	**6,534**	**140.1**	**134.3**	**274.3**	4.70
MNIST	E-FNN	784|784|784|784|784|56|10	2,506,290	1,004.9	8,799.6	9,804.5	5.18
	KDL-NN	$(28,28)\|^2(28,28)\|^2(28,28)\|^2(5,2)$	13,068	236.6	288.0	524.5	**4.04**
	E-FNN	784|1568|784|1568|784|112|10	5,011,002	1,627.1	17,137.8	18,764.8	4.76

Table 3: Network architectures for each example, including total number of trainable parameters, 20-Epoch training time (s) divided into forward propagation (→), back-propagation (←), and total training time, and test data loss. The optimal result for each category is boldfaced without regard to rounding.

rates running for 10 epochs when increasing rank, in comparison to preset ranks ranging form 1 to 3. Results are shown for an average over 10 runs to promote smoothness.



(a) BSD (a) (b) BSD (b) (c) BF (d) MNIST

Figure 5: Figures 5a to 5d show the average test errors and timing breakdown for KDL-NNs with ranks $k = 1$, $k = 2$, $k = 3$, and adaptive rank for BSD (a and b), BF, and MNIST respectively over 10 trials.

6. Conclusions We have introduced a new neural network architecture, the KDL-NN, for use in deep learning. The architecture has been developed to exploit computational acceleration afforded by a Kronecker product representation of matrix multiplication when multiplying by large weight matrices during training. For an $m \times n$ matrix that is represent as the Kronecker product of $m_1 \times n_1$ and $m_2 \times n_2$ matrices ($m_1 m_2 = m$, $n_1 n_2 = n$), then analysis and practical evaluations have shown that when moderately large factors $m_1 \approx m_2 \approx n_1 \approx n_2$ are available for numbers of nodes, training a KDL-NN requires significantly less time compared to an FNN. In addition, we have shown on several examples that the resulting accuracy of using a KDL-NN is generally improved compared to a FNN, and seems to be comparable to essentially doubling the number of layers. However, further analysis is required to determine the extent to which this holds. In particular, our analysis does not reveal precisely what properties of the data suggest that a KDL-NN approach is effective.

Further, adding weight matrices to a KDL-NN is straightforward, but practical examples have shown that altering the learning rate when increasing the representative rank may be necessary. Since KDL-NN's provide a new framework for deep learning, there are many avenues of research that are yet to be pursued. However, this work has shown the potential benefits of adopting KDLs and provided impetus to further establish the extent to which they may prove relevant. In the supplementary documentation of this paper, we show that a higher order Kronecker Multi-Layer NN (KML-NN) is feasible, but examples we have investigated suggest that such a generalization may be less effective than the simpler KDL-NN approach.

A. KDL Back-Propagation Derivation Given a KDL-NN with $L-1$ KDL pairs and $A_R^{(1)} = X$, define

$$Z_L^{(\ell,i)} = A_R^{(\ell-1)} W_L^{(\ell,i)} + B_L^{(\ell,i)}, \qquad A_L^{(\ell,i)} = \phi_2(Z_L^{(\ell,i)}),$$
$$Z_R^{(\ell,i)} = W_R^{(\ell,i)} A_L^{(\ell,i)} + B_R^{(\ell,i)}, \qquad A_R^{(\ell)} = \sum_i \phi_1(Z_R^{(\ell,i)}).$$

Differentiation from layer L to 2, splitting the loss function $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$ with $\mathcal{L}_1 = \frac{1}{2}\left\|Y - A_R^{(L)}\right\|_F^2$, and using $\circ$ to represent element-wise multiplication produces,

$$\frac{\partial \mathcal{L}}{\partial W_R^{(\ell,i)}} = \frac{\partial \mathcal{L}_1}{\partial Z_R^{(\ell,i)}} \frac{\partial Z_R^{(\ell,i)}}{\partial W_R^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial W_R^{(\ell,i)}} = \Delta_1^{(\ell,i)} A_L^{(\ell,i)T} + \lambda W_R^{(\ell,i)}$$
$$\frac{\partial \mathcal{L}}{\partial B_R^{(\ell,i)}} = \frac{\partial \mathcal{L}_1}{\partial Z_R^{(\ell,i)}} \frac{\partial Z_R^{(\ell,i)}}{\partial B_R^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial B_R^{(\ell,i)}} = \Delta_1^{(\ell,i)} + \lambda B_R^{(\ell,i)}$$
$$\frac{\partial \mathcal{L}}{\partial W_L^{(\ell,i)}} = \frac{\partial \mathcal{L}_1}{\partial Z_L^{(\ell,i)}} \frac{\partial Z_L^{(\ell,i)}}{\partial W_L^{(\ell,i)}} = A_R^{(\ell-1)T} \Delta_2^{(\ell,i)} + \frac{\partial \mathcal{L}_2}{\partial W_L^{(\ell,i)}} + \lambda W_L^{(\ell,i)}$$
$$\frac{\partial \mathcal{L}}{\partial B_L^{(\ell,i)}} = \frac{\partial \mathcal{L}_1}{\partial Z_L^{(\ell,i)}} \frac{\partial Z_L^{(\ell,i)}}{\partial B_L^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial B_L^{(\ell,i)}} = \Delta_2^{(\ell,i)} + \lambda B_L^{(\ell,i)},$$

where we have introduced the following notation for $\ell = L-1, \ldots, 2$:

$$\Gamma^{(L+1)} \coloneqq (Y - A_R^{(L)}), \qquad \Gamma^{(\ell+1)} \coloneqq \frac{\partial \mathcal{L}_1}{\partial A_R^{(\ell)}} = \sum_i \Delta_2^{(\ell+1,i)} W_L^{(\ell+1,i)T},$$
$$\Delta_1^{(\ell,i)} \coloneqq \frac{\partial \mathcal{L}_1}{\partial A_R^{(\ell)}}, \qquad \Delta_2^{(\ell,i)} \coloneqq \frac{\partial \mathcal{L}_1}{\partial Z_R^{(\ell,i)}} \frac{\partial Z_R^{(\ell,i)}}{\partial Z_L^{(\ell,i)}} = ((W_R^{(\ell,i)})^T \Delta_1^{(\ell,i)}) \circ \phi_2'(Z_L^{(\ell,i)}),$$
$$\frac{\partial A_R^{(\ell)}}{\partial Z_R^{(\ell,i)}} = \Gamma^{(\ell+1)} \circ \phi_1'(Z_R^{(\ell,i)})$$

B. Proof of Theorem 4.1 The proof of Theorem 4.1 relies on some lemmas. Lemma B.1 below computes Lipschitz constants for the individual functions f_ℓ defined in (9). Lemmas B.2 and B.3 compute error estimates associated with KPD truncations of weight matrices, and hence leverage the foundational Kronecker product rearrangement result, Lemma 3.1. The final intermediate result, Lemma B.4, computes an error estimate for a single layer of the KPD-NN versus a corresponding FNN. Following this, the proof of Theorem 4.1 is furnished.

Lemma B.1. *Given KDL forward operation f_ℓ from (9) with c_1- and c_2-Lipschitz activation functions ϕ_1 and ϕ_2 respectively, then f_ℓ is $C^{(\ell)}$-Lipschitz, where*

$$C^{(\ell)} = c_1 c_2 \left|\theta^{(\ell)}\right|_k^2.$$

Proof. Since ϕ_1 and ϕ_2 are Lipschitz, given inputs X_1 and X_2,

$$\begin{aligned}
&\|f_\ell(X_1) - f_\ell(X_2)\|_F \\
&\quad\le c_1 \left\| \sum_{i=1}^{k} W_R^{(\ell,i)} \phi_2\left(X_1 W_L^{(\ell,i)} + B_L^{(\ell,i)}\right) - \sum_{i=1}^{k} W_R^{(\ell,i)} \phi_2\left(X_2 W_L^{(\ell,i)} + B_L^{(\ell,i)}\right) \right\|_F \\
&\quad\le c_1 \sum_{i=1}^{k} \left\| W_R^{(\ell,i)} \phi_2\left(X_1 W_L^{(\ell,i)} + B_L^{(\ell,i)}\right) - W_R^{(\ell,i)} \phi_2\left(X_2 W_L^{(\ell,i)} + B_L^{(\ell,i)}\right) \right\|_F \\
&\quad\le c_1 \sum_{i=1}^{k} \left\| W_R^{(\ell,i)} \right\|_F \left\| \phi_2\left(X_1 W_L^{(\ell,i)} + B_L^{(\ell,i)}\right) - \phi_2\left(X_2 W_L^{(\ell,i)} + B_L^{(\ell,i)}\right) \right\|_F \\
&\quad\le c_1 c_2 \sum_{i=1}^{k} \left\| W_R^{(\ell,i)} \right\|_F \left\| X_1 W_L^{(\ell,i)} + B_L^{(\ell,i)} - X_2 W_L^{(\ell,i)} - B_L^{(\ell,i)} \right\|_F \\
&\quad\le c_1 c_2 \sum_{i=1}^{k} \left\| W_R^{(\ell,i)} \right\|_F \|X_1 - X_2\|_F \left\| W_L^{(\ell,i)} \right\|_F \\
&\quad= \left(c_1 c_2 \left| \theta^{(\ell)} \right|_k^2 \right) \|X_1 - X_2\|_F
\end{aligned}$$

□

Lemma B.2. *Suppose fully connected FNN output $\boldsymbol{y}$ with matrix reshaping Y has L layers and output at layer $\ell \in (2, L)$ with parameters $\tilde{\theta}$, activation function $\phi = \phi_1$, layer input $\boldsymbol{a}^{(n-1)}$ with $\boldsymbol{a}^{(1)} = \boldsymbol{x}_m$ for training pair $(\boldsymbol{x}_m, \boldsymbol{y}_m)$ with matrix reshapings X_m and Y_m, then there exists θ, and activation function ϕ_2 such that for full-rank KDL-NN output Y_r^κ with L layer pairs and layer pair $\ell \in (2, L)$ given by* (**??**),

$$\underset{\theta, \phi_2}{\arg\min} \|Y_m - Y_r^\kappa(X_m)\|_F^2 \le \|Y_m - Y(X_m)\|_F^2 \tag{13}$$

Proof. First note that for KPD $W^{(\ell)} = \sum_{i=1}^{r} L^{(\ell,i)T} \otimes R^{(\ell,i)}$, setting $W_L^{(\ell,i)} = L^{(\ell,i)}$, $W_R^{(\ell,i)} = R^{(\ell,i)}$, $B_L^{(\ell,i)} = 0$, $\operatorname{vec}\left(\sum_{i=1}^{r} B_R^{(\ell,i)}\right) = \boldsymbol{b}$, and using ϕ_2 as the linear activation function, then $A_R^{(\ell)}$ is a reshaping of $\boldsymbol{a}^{(\ell)}$ for $\ell \in (2, L)$, and $Y_r^\kappa(X_m) = Y(X_m)$. Thus, the general result holds. □

Lemma B.3. *Under assumptions of Lemma B.2, setting $k < r$, and for KPD at layer ℓ $\sum_{i=1}^{r} L^{(\ell,i)T} \otimes R^{(\ell,i)} = W^{(\ell)}$, then*

$$\left\| \left(\sum_{i=1}^{k} L^{(\ell,i)T} \otimes R^{(\ell,i)} \right) \boldsymbol{a}^{(\ell-1)} - W^{(\ell)} \boldsymbol{a}^{(\ell-1)} \right\|_2 \le \epsilon^{(\ell,k)} \left\| \boldsymbol{a}^{(\ell-1)} \right\|_2,$$

where $\epsilon^{(\ell,k)} = \left(\sum_{i=k+1}^{r} \sigma_i^{(\ell)2}\right)^{\frac{1}{2}}$ for $\sigma_i^{(\ell)}$ as the i^{th} singular value of $\mathcal{R}(W^{(\ell)})$.

Proof.

$$\begin{aligned}
&\left\| \left(\sum_{i=1}^{k} L^{(\ell,i)T} \otimes R^{(\ell,i)} \right) \boldsymbol{a}^{(\ell-1)} - W^{(\ell)} \boldsymbol{a}^{(\ell-1)} \right\|_2^2 \\
&\qquad = \left\| \left(\sum_{i=1}^{k} L^{(\ell,i)T} \otimes R^{(\ell,i)} - W^{(\ell)} \right) \boldsymbol{a}^{(\ell-1)} \right\|_2^2 \\
&\qquad \leq \left\| \left(\sum_{i=1}^{k} L^{(\ell,i)T} \otimes R^{(\ell,i)} - W^{(\ell)} \right) \right\|_2^2 \left\| \boldsymbol{a}^{(\ell-1)} \right\|_2^2 \\
&\qquad = \sum_{i=k+1}^{r} \sigma_i^{(\ell)2} \left\| \boldsymbol{a}^{(\ell-1)} \right\|_2^2,
\end{aligned}$$

where the final equality holds by Lemma 3.1. □

Lemma B.4. *Under assumptions of Lemma B.3, and for c_1-Lipschitz activation functions ϕ and ϕ_1, and layer operator $f_\ell = \left(\phi_1 \circ h_{\theta_R^{(\ell)}} \circ \phi_2 \circ h_{\theta_L^{(\ell)}} \right)$, then there exists θ, and activation function ϕ_2 such that*

$$\arg\min_{\theta, \phi_2} \left\| f_\ell \left(A^{(\ell-1)} \right) - A^{(\ell)} \right\|_F \leq c_1 \epsilon^{(\ell,k)} \left\| A^{(\ell-1)} \right\|_F,$$

where $\epsilon^{(\ell,k)} = \left(\sum_{i=k+1}^{r} \sigma_i^{(\ell)2} \right)^{\frac{1}{2}}$ for $\sigma_i^{(\ell)}$ as the i^{th} singular value of $\mathcal{R}(W^{(\ell)})$.

Proof. Setting $B_L^{(\ell,i)} = 0$, $\sum_{i=1}^{k} \operatorname{vec}(B_R^{(\ell,i)}) = \boldsymbol{b}^{(\ell)}$, $W_L^{(\ell,i)} = L^{(\ell,i)}$, $W_R^{(\ell,i)} = R^{(\ell,i)}$, and choosing ϕ_2 as the linear activation function yields

$$\operatorname{vec} \left(\sum_{i=1}^{k} W_R^{(\ell,i)} \left(A^{(\ell-1)} W_L^{(\ell,i)} + B_L^{(\ell,i)} \right) + B_R^{(\ell,i)} \right) \quad = \quad \left(\sum_{i=1}^{k} L^{(\ell,i)T} \otimes R^{(\ell,i)} \right) \boldsymbol{a}^{(\ell-1)} \; + \; \boldsymbol{b}^{(\ell)},$$

and

$$\begin{aligned}
&\left\| \phi \left(\left(\sum_{i=1}^{k} L^{(\ell,i)T} \otimes R^{(\ell,i)} \right) \boldsymbol{a}^{(\ell-1)} + \boldsymbol{b}^{(\ell)} \right) - \phi \left(W^{(\ell)} \boldsymbol{a}^{(\ell-1)} + \boldsymbol{b}^{(\ell)} \right) \right\|_2 \\
&\qquad \leq c_1 \left\| \left(\sum_{i=1}^{k} L^{(\ell,i)T} \otimes R^{(\ell,i)} \right) \boldsymbol{a}^{(\ell-1)} - W^{(\ell)} \boldsymbol{a}^{(\ell-1)} \right\|_2,
\end{aligned}$$

since ϕ is c_1-Lipschitz. Now applying Lemma B.3 and reshaping into matrix format, the general result holds. □

Proof of Theorem 4.1. Define the error at layer ℓ by $E^{(\ell)} = \left\| A_R^{(\ell)} - A^{(\ell)} \right\|_F$, and error from applying KDL forward operator f_ℓ by $e^{(\ell)} = \left\| f_\ell \left(A^{(\ell-1)} \right) - A^{(\ell)} \right\|_F$. Then by Lemma B.4 and by definition of $f_1(0)$,

$$E^{(2)} = e^{(2)} \leq c_1 \epsilon^{(2,k)} \left\| X \right\|_F = c_1 \epsilon^{(2,k)} \left\| f_1(0) \right\|_F,$$

since $A^{(1)} = A_R^{(1)} = X$, and

$$\begin{aligned}
e^{(\ell)} &\leq c_1 \epsilon^{(\ell,k)} \left\| A^{(\ell-1)} \right\|_F, \\
&= c_1 \epsilon^{(\ell,k)} \left\| f_{\ell-1}\left(A^{(\ell-2)}\right) - f_{\ell-1}(0) + f_{\ell-1}(0) \right\|_F \\
&\leq c_1^2 \epsilon^{(\ell,k)} \left\| A^{(\ell-2)} - 0 \right\|_F + c_1 \epsilon^{(\ell,k)} \left\| f_{\ell-1}(0) \right\|_F \\
&= c_1^2 \epsilon^{(\ell,k)} \left\| f_{\ell-2}\left(A^{(\ell-3)}\right) - f_{\ell-2}(0) + f_{\ell-2}(0) \right\|_F + c_1 \epsilon^{(\ell,k)} \left\| f_{\ell-1}(0) \right\|_F \\
&\vdots \\
&\leq \sum_{i=1}^{n-1} c_1^{n-i} \left\| f_i(0) \right\|_F.
\end{aligned}$$

Further,

$$\begin{aligned}
E^{(\ell)} &= \left\| A_R^{(\ell)} - f_\ell\left(A^{(\ell-1)}\right) + f_\ell\left(A^{(\ell-1)}\right) - A^{(\ell)} \right\|_F \\
&\leq \left\| A_R^{(\ell)} - f_\ell\left(A^{(\ell-1)}\right) \right\|_F + e^{(\ell)}.
\end{aligned}$$

By Lemma B.1,

$$\begin{aligned}
\left\| f_\ell\left(A_R^{(\ell-1)}\right) - f_\ell\left(A^{(\ell-1)}\right) \right\|_F &\leq C^{(\ell)} \left\| A_R^{(\ell-1)} - A^{(\ell-1)} \right\|_F \\
&= C^{(\ell)} E^{(\ell-1)}
\end{aligned}$$

Thus,

$$E^{(\ell)} \leq C^{(\ell)} E^{(\ell-1)} + \epsilon^{(\ell,k)} \sum_{i=1}^{\ell-1} c_1^{\ell-i} \left\| f_i(0) \right\|_F,$$

and

$$
\begin{aligned}
E^{(L)} &\leq C^{(L)} \left(C^{(L-1)} \left(\cdots \left(c_1 \epsilon^{(2,k)} C^{(3)} \left\| f_1(0) \right\|_F + \cdots \right) \cdots \right) + \cdots \right) \\
&\quad + \epsilon^{(L,k)} \sum_{i=1}^{L-1} c_1^{L-i} \left\| f_i(0) \right\|_F \\
&= \sum_{i=2}^{L} \epsilon^{(i,k)} \left(\prod_{j=i+1}^{L} C^{(j)} \right) \sum_{k=1}^{i-1} c_1^{i-k} \left\| f_k(0) \right\|_F \\
&= \sum_{i=2}^{L} \epsilon^{(i,k)} \left(\prod_{j=i+1}^{L} c_1 c_2 \left| \theta^{(j)} \right|_k^2 \right) \sum_{k=1}^{i-1} c_1^{i-k} \left\| f_k(0) \right\|_F \\
&= \sum_{i=2}^{L} \epsilon^{(i,k)} \left(\prod_{j=i+1}^{L} c_2 \left| \theta^{(j)} \right|_k^2 \right) c_1^{(L-i)} \sum_{k=1}^{i-1} c_1^{i-k} \left\| f_k(0) \right\|_F \\
&= \sum_{i=2}^{L} \epsilon^{(i,k)} \left(\prod_{j=i+1}^{L} c_2 \left| \theta^{(j)} \right|_k^2 \right) \sum_{k=1}^{i-1} c_1^{L-k} \left\| f_k(0) \right\|_F .
\end{aligned}
$$

Finally, given $\phi_2(X) = X$, then $c_2 = 1$. □

References

[1] B. Adcock and N. Dexter, *The gap between theory and practice in function approximation with deep neural networks*, SIAM Journal on Mathematics of Data Science, 3 (2021), pp. 624–655.

[2] A. Aghasi, A. Abdi, and J. Romberg, *Fast Convex Pruning of Deep Neural Networks*, SIAM Journal on Mathematics of Data Science, 2 (2020), pp. 158–188, https://doi.org/10.1137/19M1246468, https://epubs.siam.org/doi/abs/10.1137/19M1246468 (accessed 2022-02-09).

[3] K. Buza, *Feedback prediction for blogs*, in Data analysis, machine learning and knowledge discovery, Springer, 2014, pp. 145–152.

[4] A. Chkifa, N. Dexter, H. Tran, and C. Webster, *Polynomial approximation via compressed sensing of high-dimensional functions on lower sets*, Mathematics of Computation, 87 (2018), pp. 1415–1450.

[5] H. Fanaee-T and J. Gama, *Event labeling combining ensemble detectors and background knowledge*, Progress in Artificial Intelligence, (2013), pp. 1–15, https://doi.org/10.1007/s13748-013-0040-3, [WebLink].

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, JHU Press, fourth ed., 2013, http://www.cs.cornell.edu/cv/GVL4/golubandvanloan.htm.

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

[8] M. G. A. Hameed, M. S. Tahaei, A. Mosleh, and V. P. Nia, *Convolutional neural network compression through generalized kronecker product decomposition*, arXiv preprint arXiv:2109.14710, (2021).

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, arXiv:1207.0580 [cs], (2012), http://arxiv.org/abs/1207.0580. arXiv: 1207.0580.

[10] A. D. Jagtap, Y. Shin, K. Kawaguchi, and G. E. Karniadakis, *Deep kronecker neural networks: A general framework for neural networks with adaptive activation functions*, arXiv preprint arXiv:2105.09513, (2021).

[11] A. Labach, H. Salehinejad, and S. Valaee, *Survey of Dropout Methods for Deep Neural Networks*, arXiv:1904.13310 [cs], (2019), http://arxiv.org/abs/1904.13310. arXiv: 1904.13310.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[13] J. Martens and R. Grosse, *Optimizing neural networks with Kronecker-factored approximate curvature*, in International conference on machine learning, PMLR, 2015, pp. 2408–2417.

[14] Y. Movshovitz-Attias and E. Eban, *Weight compression for deep networks using kronecker products*, Technical Disclosure Commons, (2018).

[15] K. Scaman and A. Virmaux, *Lipschitz regularity of deep neural networks: analysis and efficient estimation*, arXiv preprint arXiv:1805.10965, (2018).

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research, 15 (2014), pp. 1929–1958, http://jmlr.org/papers/v15/srivastava14a.html.

[17] C. F. Van Loan, *The ubiquitous kronecker product*, Journal of computational and applied mathematics, 123 (2000), pp. 85–100.

[18] C. F. Van Loan and N. Pitsianis, *Approximation with Kronecker products*, in Linear algebra for large scale and real-time applications, Springer, 1993, pp. 293–314.

[19] S. Zhou, J.-N. Wu, Y. Wu, and X. Zhou, *Exploiting local structures with the Kronecker layer in convolutional networks*, arXiv preprint arXiv:1512.09194, (2015).

[20] M. Zhu and S. Gupta, *To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression*, in 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings, OpenReview.net, 2018, https://openreview.net/forum?id=Sy1iIDkPM (accessed 2022-02-10).

Supplementary Materials: Dimensionality Reduction in Deep Learning via Kronecker Multi-layer Architectures*

Jarom D. Hogue† Robert M. Kirby‡ Akil Narayan§

April 12, 2022

S. High Order Kronecker Multi-Layer A Kronecker multi-layer (KML) approach is explained here. Given a KDL, consider the refinements $W_L^{(\ell,i)} = \sum_{j=1}^r W_2^{(\ell,j)T} \otimes W_1^{(\ell,j)}$ and $W_R^{(\ell,i)} = \sum_{j=1}^r W_3^{(\ell,j)} \otimes W_4^{(\ell,j)T}$. Then for column $\boldsymbol{a}_i$ of A, $(A^T W_L^{(\ell,i)})^T = \sum_{j=1}^r \left[\begin{array}{ccc} (W_1^{(\ell,j)T} \otimes W_2^{(\ell,j)})\boldsymbol{a}_1 & \cdots & (W_1^{(\ell,j)T} \otimes W_2^{(\ell,j)})\boldsymbol{a}_p \end{array} \right]$, and with a slight abuse of notation, $W_R^{(\ell,i)} A = \sum_{j=1}^r \left[\begin{array}{ccc} (W_3^{(\ell,j)T} \otimes W_4^{(\ell,j)})\boldsymbol{a}_1 & \cdots & (W_3^{(\ell,j)T} \otimes W_4^{(\ell,j)})\boldsymbol{a}_p \end{array} \right]$. KP multiplication operations with $A_i = (mat)(\boldsymbol{a}_i$ are then implemented as $W_2^{(\ell,j)})A_i W_1^{(\ell,j)}$ and $W_2^{(\ell,j)})A_i W_1^{(\ell,j)}$. This even split into 4 multi-layers will be referred to in the node configuration by a refinement using parenthesis. i.e. a KDL given by $(28, 28)$ could be refined into a configuration given by $((7, 4), (7, 4))$.

Adding activation functions and bias terms, the multi-layers can be written as

$$
\begin{aligned}
Z_1^{(\ell,i,j)} &= A_{4(j,:)}^{(\ell-1)} W_1^{(\ell,i)} + B_1^{(\ell,i)}, & A_1^{(\ell,i,j)} &= \phi(Z_1^{(\ell,i,j)}), \\
Z_2^{(\ell,i,j)} &= W_2^{(\ell,i)} A_1^{(\ell,i,j)} + B_2^{(\ell,i)}, & A_{2(j,:)}^{(\ell,i)} &= \phi(\mathrm{vec}(Z_2^{(\ell,i,j)})^T), \\
Z_3^{(\ell,i,j)} &= A_{2(:,j)}^{(\ell,i)} W_3^{(\ell,i)} + B_3^{(\ell,i)}, & A_3^{(\ell,i,j)} &= \phi(Z_3^{(\ell,i,j)}), \\
Z_4^{(\ell,i,j)} &= W_4^{(\ell,i)} A_3^{(\ell,i,j)} + B_4^{(\ell,i)}, & A_{4(:,j)}^{(\ell)} &= \sum_i \phi(\mathrm{vec}(Z_4^{(\ell,i,j)})),
\end{aligned}
$$

where MATLAB style notation is used in subscripts to differentiate between rows or columns being reshaped into matrix form.

*This research was sponsored by ARL under Cooperative Agreement Number W911NF-12-2-0023. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ARL or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. The first and third authors are partially supported by NSF DMS-1848508 and AFOSR FA9550-20-1-0338.

†Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT(jdhogue@sci.utah.edu).

‡Scientific Computing and Imaging Institute and School of Computing, University of Utah, Salt Lake City, UT(kirby@cs.utah.edu).

§Scientific Computing and Imaging Institute and Department of Mathematics, University of Utah, Salt Lake City, UT(akil@sci.utah.edu).

Back-propagation follows with

$$\Gamma^{(L+1,j)} := (Y - A_4^{(L)})$$

$$\Gamma^{(\ell+1,j)} := \frac{\partial \mathcal{L}_1}{\partial A_{4(:,j)}^{(\ell)}} = \sum_i \Delta_1^{(\ell+1,i,j)} W_1^{(\ell+1,i)T}, \qquad \ell = L-1, \cdots 2$$

$$\Delta_4^{(\ell,i,j)} := \frac{\partial \mathcal{L}_1}{\partial A_{4(:,j)}^{(\ell)}} \frac{\partial A_{4(:,j)}^{(\ell)}}{\partial Z_4^{(\ell,i,j)}} = \sum_j \Gamma^{(\ell+1,j)} \circ \phi'(Z_4^{(\ell,i,j)})$$

$$\frac{\partial \mathcal{L}}{\partial W_4^{(\ell,i)}} = \sum_j \frac{\partial \mathcal{L}_1}{\partial Z_4^{(\ell,i,j)}} \frac{\partial Z_4^{(\ell,i,j)}}{\partial W_4^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial W_4^{(\ell,i)}} = \sum_j \Delta_4^{(\ell,i,j)} A_3^{(\ell,i,j)T} + \lambda W_4^{(\ell,i)}$$

$$\frac{\partial \mathcal{L}}{\partial B_4^{(\ell,i)}} = \sum_j \frac{\partial \mathcal{L}_1}{\partial Z_4^{(\ell,i,j)}} \frac{\partial Z_4^{(\ell,i,j)}}{\partial B_4^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial B_4^{(\ell,i)}} = \sum_j \Delta_4^{(\ell,i,j)} + \lambda B_4^{(\ell,i)}$$

$$\Delta_3^{(\ell,i,j)} := \frac{\partial \mathcal{L}_1}{\partial Z_4^{(\ell,i,j)}} \frac{\partial Z_4^{(\ell,i,j)}}{\partial Z_3^{(\ell,i,j)}} = ((W_4^{(\ell,i)})^T \Delta_4^{(\ell,i,j)}) \circ \phi'(Z_3^{(\ell,i,j)})$$

$$\frac{\partial \mathcal{L}}{\partial W_3^{(\ell,i)}} = \sum_j \frac{\partial \mathcal{L}_1}{\partial Z_3^{(\ell,i,j)}} \frac{\partial Z_3^{(\ell,i,j)}}{\partial W_3^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial W_3^{(\ell,i)}} = \sum_j A_2^{(\ell,i,j)T} \Delta_3^{(\ell,i,j)} + \lambda W_3^{(\ell,i)}$$

$$\frac{\partial \mathcal{L}}{\partial B_3^{(\ell,i)}} = \sum_j \frac{\partial \mathcal{L}_1}{\partial Z_3^{(\ell,i,j)}} \frac{\partial Z_3^{(\ell,i,j)}}{\partial B_3^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial B_3^{(\ell,i)}} = \sum_j \Delta_3^{(\ell,i,j)} + \lambda B_3^{(\ell,i)}$$

$$\Delta_{2(p,q)}^{(\ell,i,j)} := \frac{\partial \mathcal{L}_1}{\partial Z_{3(j,q)}^{(\ell,i,p)}} \frac{\partial Z_{3(j,q)}^{(\ell,i,p)}}{\partial Z_{2(p,q)}^{(\ell,i,j)}} = (\Delta_{3(j,q)}^{(\ell,i,p)} W_{3(p,q)}^{(\ell,i)T}) \circ \phi'(Z_{2(p,q)}^{(\ell,i,j)})$$

$$\frac{\partial \mathcal{L}}{\partial W_2^{(\ell,i)}} = \sum_j \frac{\partial \mathcal{L}_1}{\partial Z_2^{(\ell,i,j)}} \frac{\partial Z_2^{(\ell,i,j)}}{\partial W_2^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial W_2^{(\ell,i)}} = \sum_j \Delta_2^{(\ell,i,j)} A_1^{(\ell,i,j)T} + \lambda W_2^{(\ell,i)}$$

$$\frac{\partial \mathcal{L}}{\partial B_2^{(\ell,i)}} = \sum_j \frac{\partial \mathcal{L}_1}{\partial Z_2^{(\ell,i,j)}} \frac{\partial Z_2^{(\ell,i,j)}}{\partial B_2^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial B_4^{(\ell,i)}} = \sum_j \Delta_2^{(\ell,i,j)} + \lambda B_2^{(\ell,i)}$$

$$\Delta_1^{(\ell,i,j)} := \frac{\partial \mathcal{L}_1}{\partial Z_2^{(\ell,i,j)}} \frac{\partial Z_2^{(\ell,i,j)}}{\partial Z_1^{(\ell,i,j)}} = \sum_j ((W_2^{(\ell,i)})^T \Delta_2^{(\ell,i,j)}) \circ \phi'(Z_1^{(\ell,i,j)})$$

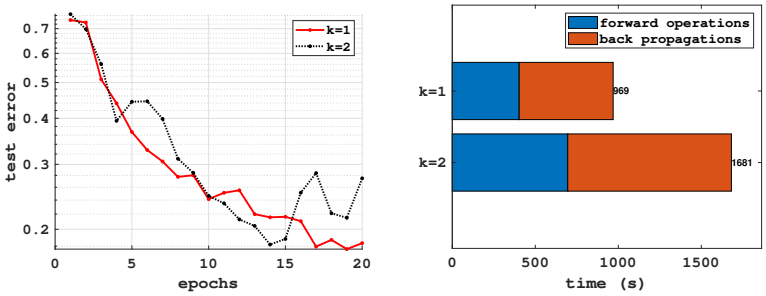
$$\frac{\partial \mathcal{L}}{\partial W_1^{(\ell,i)}} = \sum_j \frac{\partial \mathcal{L}_1}{\partial Z_1^{(\ell,i,j)}} \frac{\partial Z_1^{(\ell,i,j)}}{\partial W_1^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial W_1^{(\ell,i)}} = \sum_j A_2^{(\ell,i,j)T} \Delta_1^{(\ell,i,j)} + \lambda W_1^{(\ell,i)}$$

$$\frac{\partial \mathcal{L}}{\partial B_1^{(\ell,i)}} = \sum_j \frac{\partial \mathcal{L}_1}{\partial Z_1^{(\ell,i,j)}} \frac{\partial Z_1^{(\ell,i,j)}}{\partial B_1^{(\ell,i)}} + \frac{\partial \mathcal{L}_2}{\partial B_1^{(\ell,i)}} = \sum_j \Delta_1^{(\ell,i,j)} + \lambda B_1^{(\ell,i)}$$

In practice, viable training has required separate weights and biases for each layer of the split, i.e.

$$
\begin{aligned}
Z_1^{(\ell,i,j)} &= A_{4(j,:)}^{(\ell-1)} W_1^{(\ell,i,j)} + B_1^{(\ell,i,j)}, & A_1^{(\ell,i,j)} &= \phi(Z_1^{(\ell,i,j)}),\\
Z_2^{(\ell,i,j)} &= W_2^{(\ell,i,j)} A_1^{(\ell,i,j)} + B_2^{(\ell,i,j)}, & A_{2(j,:)}^{(\ell,i)} &= \phi(\mathrm{vec}(Z_2^{(\ell,i,j)})^T),\\
Z_3^{(\ell,i,j)} &= A_{2(:,j)}^{(\ell,i)} W_3^{(\ell,i,j)} + B_3^{(\ell,i,j)}, & A_3^{(\ell,i,j)} &= \phi(Z_3^{(\ell,i,j)}),\\
Z_4^{(\ell,i,j)} &= W_4^{(\ell,i,j)} A_3^{(\ell,i,j)} + B_4^{(\ell,i,j)}, & A_{4(:,j)}^{(\ell)} &= \sum_i \phi(\mathrm{vec}(Z_4^{(\ell,i,j)})).
\end{aligned}
$$

Results are shown in Figure 6 using an even 4-split KML-NN on MNIST with node configuration $N = \{((7,4),(7,4)),((7,4),(7,4)),((7,4),(7,4)),((5,1),(2,1))\}$. Similar to using KDL-NN on BSD (a), using this KML-NN on MNIST with these small values results in an overall increase in time, and further analysis on larger sets is still required to determine the benefits of adopting higher order KMLs.



(a) MNIST (b) MNIST

Figure 6: Figures 6a and 6b show the test errors and timing breakdowns for a KML-NN with 4 multi-layers with Kronecker ranks 1 and 2 for MNIST.