Learning Compressed Transforms with Low Displacement Rank

Anna T. Thomas*,†, Albert Gu*,†, Tri Dao†, Atri Rudra‡, and Christopher Ré†

†Department of Computer Science, Stanford University
‡Department of Computer Science and Engineering, University at Buffalo, SUNY

{*thomasat,albertgu,trid}@stanford.edu, atri@buffalo.edu, chrismre@cs.stanford.edu

September 20, 2018

Abstract

The low displacement rank (LDR) framework for structured matrices represents a matrix through two displacement operators and a low-rank residual. Existing use of LDR matrices in deep learning has applied fixed displacement operators encoding forms of shift invariance akin to convolutions. We introduce a rich class of LDR matrices with much more general displacement operators, and explicitly learn over both the operators and the low-rank component. This class generalizes several previous constructions while preserving compression and efficient computation. We prove bounds on the the VC dimension of multi-layer neural networks with structured weight matrices and show empirically that our compact parameterization can reduce the sample complexity of learning. When replacing weight layers in fully-connected, convolutional, and recurrent neural networks for image classification and language modeling tasks, our new classes exceed the accuracy of existing compression approaches, and on some tasks even outperform fully-connected layers while using more than 20X fewer parameters.

1 Introduction

Recent years have seen a surge of interest in structured representations for deep learning, motivated by achieving compression and acceleration while maintaining generalization properties. A popular approach for learning compact models involves constraining the weight matrices to exhibit some form of dense but compressible structure and learning directly over the parameterization of this structure. Examples of structures explored for the weight matrices of deep learning pipelines include low-rank matrices, low-distortion projections [44], (block-)circulant matrices [8, 17], Toeplitz-like matrices [40], and constructions derived from Fourier-related transforms [32]. Though they confer significant storage and computation benefits, these constructions lack modeling capacity and tend to underperform general fully-connected layers in deep learning. This raises the question of whether broader classes of structured matrices can achieve superior downstream performance while retaining compression guarantees.

Our approach leverages the low displacement rank (LDR) framework (Section 2), which encodes structure through two sparse displacement operators and a low-rank remainder term [23]. Previous work studying neural networks with LDR weight matrices assumes fixed displacement operators and learns only over the remainder [10, 45]. The only case attempted in practice that explicitly employs the LDR framework uses fixed operators encoding shift invariance, producing weight matrices which were found to outperform several other compression approaches [10]. Unlike previous work, we consider learning the displacement operators jointly with the low-rank residual. Building upon recent progress on structured dense matrix-vector multiplication [14], we introduce a much more general class of LDR matrices and develop practical
algorithms for using these matrices in deep learning architectures. We show that the resulting class of matrices subsumes many previously used structured layers, including constructions that did not explicitly use the LDR framework [17, 32]. When compressing fully-connected weight matrices in single hidden layer, convolutional, and recurrent neural networks, we empirically demonstrate improved accuracy over existing approaches. Furthermore, on multiple tasks our constructions achieve higher accuracy than even fully-connected (FC) layers while using an order of magnitude fewer parameters.

To shed light on the empirical success of LDR matrices in machine learning, we draw connections to recent work on learning equivariant representations, and hope to motivate further investigations of this link. Notably, many successful previous methods for compression apply classes of matrices similar to convolutions [8, 17, 40]; while their explicit aim is to accelerate training and reduce memory costs, this constraint implicitly encodes a shift-invariant structure that is well-suited for image and audio data. However, this requires crucial underlying assumptions about latent invariances in the data, which may not be precisely known in advance. We observe that the LDR construction enforces a natural notion of approximate equivariance to transformations governed by the displacement operators, suggesting that our approach of learning the operators allows for modeling and learning more general latent structures in data.

Despite their increased expressiveness, our new classes retain the storage and computational benefits of conventional structured representations. Our construction provides guaranteed compression (from quadratic to linear parameters) and matrix-vector multiplication algorithms that are quasi-linear in the number of parameters. We additionally provide the first analysis of the sample complexity of learning neural networks with LDR weight matrices, which extends to low-rank, Toeplitz-like and other previously explored fixed classes of LDR matrices. More generally, our analysis applies to structured matrices whose parameters can interact multiplicatively with high degree. We prove that the class of neural networks constructed from these matrices retains VC dimension almost linear in the number of parameters, which implies that LDR matrices with learned displacement operators are still efficiently recoverable from data. This is consistent with our empirical results, which suggest that constraining weight layers to our broad class of LDR matrices reduces the sample complexity of learning compared to fully-connected weights.

We provide a detailed review of previous work and connections to our approach in Appendix B.

Summary of contributions

- We introduce a rich class of LDR matrices where the displacement operators are explicitly learned from data, and provide multiplication algorithms implemented in PyTorch (Section 3).

- We prove that the VC dimension of multi-layer neural networks with LDR weight matrices, which encompasses a broad class of previously explored approaches including the low-rank and Toeplitz-like classes, is quasi-linear in the number of parameters (Section 4).

- We empirically demonstrate that our construction improves quality when compressing FC layers in fully-connected, convolutional, and recurrent neural networks compared to previous compression approaches, and on some tasks can even outperform FC layers (Section 5).

2 Background: displacement rank

The generic term structured matrix refers to an \( m \times n \) matrix that can be represented in much fewer than \( mn \) parameters, and admits fast operations such as matrix-vector multiplication. The displacement rank approach represents a structured matrix \( M \in \mathbb{R}^{m \times n} \) through displacement operators \((A \in \mathbb{R}^{m \times m}, B \in \mathbb{R}^{n \times n})\) defining a linear map \( \nabla_{A,B} : M \mapsto AM - MB \) on matrices, and a remainder \( R \), so that

\[
AM - MB = R
\]

then \( M \) can be manipulated solely through the compressed representation \((A, B, R)\). The rank of \( \nabla_{A,B} M \) is called the displacement rank of \( M \) w.r.t. \((A, B)\).
The displacement approach was originally introduced to describe the Toeplitz-like matrices, which are not perfectly Toeplitz but still have shift-invariant structure [23]. These have LDR with respect to shift/cycle operators. A standard formulation uses $A = Z_1, B = Z_{-1}$, where $Z_f = \begin{bmatrix} 0_{1 \times (n-1)} & f \\ I_{n-1} & 0_{(n-1) \times 1} \end{bmatrix}$ denotes the matrix with 1 on the subdiagonal and $f$ in the top-right corner. In the context of machine learning, these have been analyzed for neural networks, sequence models [30], and kernel approximation [10]. They are similar to convolutions due to the shift invariance, and in several cases have performed significantly better than competing compressed approaches [40]. Figure 1 illustrates the displacement(1) for a Toeplitz matrix, showing how the matrix’s shift invariant structure leads to a remainder of rank at most 2.

$$M = \begin{bmatrix} a_0 & a_1 & \cdots & a_{n-1} \\ a_{-1} & a_0 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{-n+1} & \cdots & a_{-1} & a_0 \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_0 & a_1 & \cdots & a_{n-1} \\ a_{-1} & a_0 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{-n+1} & \cdots & a_{-1} & a_0 \end{bmatrix}$$

Figure 1: Displacement equation for a Toeplitz matrix with respect to shift operators $Z_1, Z_{-1}$. Empty entries are zero.

A few distinct classes of useful matrices are known to satisfy a displacement property: the classic types are the Toeplitz-, Hankel-, Vandermonde-, and Cauchy-like matrices [23], which are ubiquitous in other disciplines [35]. These classes have fixed operators consisting of diagonal or shift matrices, and LDR properties have traditionally been analyzed in detail only for these special cases. Nonetheless, a few elegant properties hold for generic operators, stating that certain combinations of (and operators on) LDR matrices preserve low displacement rank. We call these closure properties, and introduce an additional block closure property that is related to convolutional filter channels (Section 5.2).

We use the notation $D_{r,A,B}$ to refer to the matrices of displacement rank $\leq r$ with respect to $(A,B)$.

**Proposition 1.** LDR matrices are closed under the following operations:

(a) **Transpose/Inverse** If $M \in D_{r,A,B}$, then $M^T \in D_{s,B^T,A^T}$ and $M^{-1} \in D_{r,A,B}$.

(b) **Sum** If $M \in D_{r,A,B}$ and $N \in D_{s,A,B}$, then $M + N \in D_{r+s,A,B}$.

(c) **Product** If $M \in D_{r,A,B}$ and $N \in D_{s,B,C}$, then $MN \in D_{r+s,A,C}$.

(d) **Block** Let $M_{i,j}$ satisfy $M_{i,j} \in D_{r,A,B}$, for $i = 1 \ldots k, j = 1 \ldots \ell$. Then the $k \times \ell$ block matrix $(M_{i,j})_{ij}$ has displacement rank $r k \ell$.

Proposition 1 is proved in Appendix C.

3 Learning displacement operators

We consider two classes of new displacement operators. These operators are fixed to be matrices with particular sparsity patterns, where the entries are treated as learnable parameters.

The first consists of subdiagonal operators: $A_{i+1,i}$ are the only possible non-zero entries (we additionally allow the corner $A_{0,n-1}$ to be non-zero). As $Z_f$ is a special case matching this sparsity pattern, this class is the most direct generalization of Toeplitz-like matrices with learnable operators.

The second class of operators are tridiagonal matrices: $A_{i,j}$ can only be non-zero if $|i - j| \leq 1$ (additionally, the outer corners are also allowed to be non-zero). Figure 2 shows the displacement operators for the Toeplitz-like class and our more general operators.

We henceforth let LDR-SD and LDR-TD (which contains LDR-SD) denote the matrices with low displacement rank with respect to subdiagonal and tridiagonal operators, respectively.
Expressiveness The matrices we introduce can model rich structure and subsume many types of linear transformations used in machine learning. We list some of the structured matrices that have LDR with respect to tridiagonal displacement operators:

**Proposition 2.** The LDR-TD matrices contain:

(a) Toeplitz-like matrices, which themselves include many Toeplitz and circulant variants (including standard convolutional filters) [8, 17, 40].

(b) low-rank matrices.

(c) The other classic displacement structures: Hankel-like, Vandermonde-like, and Cauchy-like matrices.

(d) Orthogonal polynomial transforms, including the Discrete Fourier and Cosine Transforms.

(e) Combinations and derivatives of these classes via the closure properties (Proposition 1), including structured classes previously used in machine learning such as ACDC [32] and block circulant layers [17].

These reductions are stated more formally and proved in Appendix C.1. We also include a diagram of the structured matrix classes included by our proposed LDR-TD and LDR-SD classes in Figure 3 in Appendix C.1.

Our parametrization Given the parameters $A, B, R$, the operation that must ultimately be performed is matrix-vector multiplication by $M = \nabla^{-1}_{A,B}[R]$. Several schemes for explicitly reconstructing $M$ from its displacement parameters are known for specific cases [36, 39], but do not always apply to our general operators. Instead, we use $A, B, R$ to implicitly construct a slightly different matrix with at most double the displacement rank that is simpler to work with.

**Proposition 3.** Let $K(A,v)$ denote the $n \times n$ Krylov matrix which has $i$-th column $A^i v$. For any vectors $g_1, \ldots, g_r \in \mathbb{R}^m$ and $h_1, \ldots, h_r \in \mathbb{R}^n$, then the matrix

$$
\sum_{i=1}^r K(A, g_i) K(B^T, h_i)^T
$$

has displacement rank at most $2r$ with respect to $A^{-1}, B$.

Thus our representation stores the parameters $A, B, G, H$, where $A, B$ are either subdiagonal or tridiagonal operators (containing $n$ or $3n$ parameters), and $G, H \in \mathbb{R}^{n \times r}$. These parameters implicitly define the matrix (2), which is the LDR weight layer we use.
Figure 3: Our proposed LDR-TD and LDR-SD structured matrix classes contain a number of other approaches including Toeplitz-like [40] (and other classic displacement types, such as Hankel-like, Vandermonde-like, and Cauchy-like), low-rank [15], circulant [8], standard one-dimensional convolutional filters, and orthogonal polynomial transforms, including the Discrete Fourier and Cosine Transforms. As noted in Section 3, here we implement the near-linear time algorithms from [14] for the LDR-SD class and have made our PyTorch implementation public. Captions for each class show storage cost and number of operations needed for matrix-vector multiplication.

**Algorithms for LDR-SD**

Generic and near-linear time algorithms for matrix-vector multiplication by LDR matrices with even more general operators, including both the LDR-TD and LDR-SD classes, were recently shown to exist [14]. However, complete algorithms were not provided, as they relied on theoretical results such as the transposition principle [6] that only imply the existence of algorithms. Additionally, the recursive polynomial-based algorithms are difficult to implement efficiently. For LDR-SD, we provide explicit and complete near-linear time algorithms for multiplication by \( \mathbf{K}(\mathbf{A}, \mathbf{g}) \) and \( \mathbf{K}(\mathbf{B}^T, \mathbf{h}) \) matrices for \( i = 1, ..., r \) from Proposition 3 and then apply the standard \( O(n^2) \) matrix-vector multiplication algorithm. Efficient implementations of near-linear time algorithms for LDR-TD are an interesting area of future work.

**Theorem 1.** Define the simultaneous computation of \( k \) Fast Fourier Transforms (FFT), each with size \( m \), to be a batched FFT with total size \( km \).

Consider any subdiagonal matrix \( \mathbf{A} \in \mathbb{R}^{n \times n} \) and vectors \( \mathbf{g}, \mathbf{h} \in \mathbb{R}^n \). Then \( \mathbf{K}(\mathbf{A}, \mathbf{g})^T \) or \( \mathbf{K}(\mathbf{A}, \mathbf{g}) \) can be multiplied by any vector \( \mathbf{x} \) by computing \( 8 \log_2(n) \) batched FFTs, each of total size \( 2n \). The total number of computations is \( O(n \log^2 n) \).

These algorithms are also automatically differentiable, which we use to compute the gradients when learning. More complete descriptions of these algorithms are presented in Appendix C.
4 Theoretical properties of structured matrices

Complexity of LDR neural networks The matrices we use are unusual in that the parameters interact multiplicatively (namely in $A^i, B^i$) to implicitly define the actual layer. In contrast, fully-connected layers are linear and other structured layers, such as Fastfood and ACDC \cite{27, 32, 44}, are constant degree in their parameters. However, we can prove that this does not significantly change the learnability of our classes:

**Theorem 2.** Let $\mathcal{F}$ denote the class of neural networks with $L$ LDR layers, $W$ total parameters, and piecewise linear activations. Let $\text{sign} \mathcal{F}$ denote the corresponding classification functions, i.e. $\{x \mapsto \text{sign} f(x) : f \in \mathcal{F}\}$. The VC dimension of this class is

$$\text{VCdim}(\text{sign} \mathcal{F}) = O(LW \log W).$$

Theorem 2 matches the standard bound for unconstrained weight matrices \cite{4, 5}. This immediately implies a standard PAC-learnable guarantee \cite{42}. Theorem 2 holds for even more general activations and matrices that for example include the broad classes of \cite{14}. The proof is in Appendix D, and we empirically validate the generalization and sample complexity properties of our class in Section 5.3.

LDR as an approximately-equivariant linear map We observe that displacement rank is actually related to a line of work outside the resource-constrained learning community, specifically on building equivariant (also called covariant in some contexts) feature representations that transform in predictable ways when the input is transformed. An equivariant feature map $\Phi$ satisfies

$$\Phi(B(x)) = A(\Phi(x))$$

for transformations $A, B$ (invariance is the special case when $A$ is the identity) \cite{16, 29, 38}. This means that perturbing the input by a transformation $B$ before passing through the map $\Phi$ is equivalent to first finding the features $\Phi$ then transforming by $A$.

Intuitively, LDR matrices are a suitable choice for modeling approximately equivariant linear maps, since the residual $A\Phi - \Phi B$ of (5) has low complexity. Furthermore, approximately equivariant maps should retain the compositional properties of equivariance, which LDR satisfies via Proposition 1. For example, Proposition 1 formalizes the notion that the composition of two approximately equivariant maps is still approximately equivariant. Using this intuition, the displacement representation (1) of a matrix decomposes into two parts: the displacement operators $A, B$ define the transformations to which the model is approximately equivariant, and the low complexity remainder $R$ controls standard model capacity.

Equivariance has been used in several ways in the context of machine learning. One formulation, used for example to model ego-motions, supposes that (5) holds only approximately, and uses a fixed transformation $B$ along with pairs of data points for both sides of (5) to learn an appropriate $A$ \cite{1, 29}. Another line of work uses the representation theory formalization of equivariant maps \cite{12, 24}. We describe this formulation in more detail and show that LDR satisfies this definition as well in Proposition 4 in Appendix C.3. In contrast to previous settings, which hold one or both of $A, B$ fixed, our formulation stipulates that $\Phi$ can be uniquely determined from $A, B$, and learns the latter as part of an end-to-end model. In Section 5.4 we include a visual example of latent structure that our displacement operators learn, where they recover centering information about objects from a 2D image dataset.

5 Empirical evaluation

Overview In Section 5.1 we consider the standard setting of compressing a single hidden layer (SHL) neural network and the fully-connected (FC) layer of a CNN for image classification tasks. Following previous work \cite{7, 40}, we test on two challenging MNIST variants \cite{26}, and include two additional datasets with more realistic objects (CIFAR-10 \cite{25} and NORB \cite{28}). Since SHL models take a single channel as input, we converted CIFAR-10 to grayscale for this task. Our classes and the structured baselines are tested across different parameter budgets in order to show tradeoffs between compression and accuracy. As shown in Table 1, our methods consistently have higher test accuracy than baselines for compressed training and inference, by 3.14, 2.70, 3.55, and 3.37 accuracy points on MNIST-bg-rot, MNIST-noise, CIFAR-10, and...
NORB respectively. Additionally, to explore whether learning the displacement operators can facilitate adaptation to other domains, we replace the input-hidden weights in an LSTM for a language modeling task, and show improvements of 0.81-30.47 perplexity points compared to baselines at several parameter budgets.

In addition to experiments on replacing fully-connected layers, in Section 5.2 we also replace the convolutional layer of a simple CNN while preserving performance within 1.1 accuracy points on CIFAR-10. In Section 5.3 we consider the effect of a higher parameter budget. By increasing the rank to just 16, the LDR-SD class exceeds the accuracy of the FC layer in all datasets we tested on, for both SHL and CNN.

Appendix F includes more experimental details and protocols. Our PyTorch code is publicly available.

5.1 Compressing fully-connected layers

*Image classification* Sindhwani et al. [40] showed that for a fixed parameter budget, the Toeplitz-like class significantly outperforms several other compression approaches. Following previous experimental settings [7, 40], Table 1 compares our proposed classes to several structured baselines for compressed training and inference when compressing the hidden layer of a single hidden layer neural network. In addition to Toeplitz-like, we implement and compare to other classic LDR types, which were previously indicated as an unexplored possibility [40, 45]. We also compare to the Neural Network (NN) baseline used in [7], in which the number of hidden units is adjusted to meet the parameter budget. We also show results when compressing the FC layer of a 7-layer CNN based on LeNet in Appendix E Table 7. In Appendix E we show comparisons to additional baselines.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST-bg-rot</th>
<th>MNIST-noise</th>
<th>CIFAR-10</th>
<th>NORB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully-connected</td>
<td>44.08</td>
<td>65.15</td>
<td>46.03</td>
<td>59.83</td>
</tr>
<tr>
<td></td>
<td>622506</td>
<td>622506</td>
<td>1058826</td>
<td>1054726</td>
</tr>
<tr>
<td>LDR-TD ((r = 1))</td>
<td><strong>45.81</strong></td>
<td><strong>78.45</strong></td>
<td><strong>45.33</strong></td>
<td><strong>62.75</strong></td>
</tr>
<tr>
<td></td>
<td>14122</td>
<td>14122</td>
<td>18442</td>
<td>14342</td>
</tr>
<tr>
<td>Toeplitz-like ([40] (r = 4))</td>
<td>42.67</td>
<td>75.75</td>
<td>41.78</td>
<td>59.38</td>
</tr>
<tr>
<td></td>
<td>14122</td>
<td>14122</td>
<td>18442</td>
<td>14342</td>
</tr>
<tr>
<td>Hankel-like ((r = 4))</td>
<td>42.23</td>
<td>73.65</td>
<td>41.40</td>
<td>60.90</td>
</tr>
<tr>
<td></td>
<td>14122</td>
<td>14122</td>
<td>18442</td>
<td>14342</td>
</tr>
<tr>
<td>Vandermonde-like ((r = 4))</td>
<td>37.14</td>
<td>59.80</td>
<td>33.93</td>
<td>48.98</td>
</tr>
<tr>
<td></td>
<td>14122</td>
<td>14122</td>
<td>18442</td>
<td>14342</td>
</tr>
<tr>
<td>Low-rank ([15] (r = 4))</td>
<td>35.67</td>
<td>52.25</td>
<td>32.28</td>
<td>43.66</td>
</tr>
<tr>
<td></td>
<td>14122</td>
<td>14122</td>
<td>18442</td>
<td>14342</td>
</tr>
<tr>
<td>NN ([7])</td>
<td>38.78</td>
<td>62.60</td>
<td>37.64</td>
<td>51.67</td>
</tr>
<tr>
<td></td>
<td>14302</td>
<td>14302</td>
<td>18622</td>
<td>14426</td>
</tr>
<tr>
<td>Fastfood ([44])</td>
<td>38.13</td>
<td>63.55</td>
<td>39.64</td>
<td>59.02</td>
</tr>
<tr>
<td></td>
<td>10202</td>
<td>10202</td>
<td>13222</td>
<td>9222</td>
</tr>
<tr>
<td>Circulant ([8])</td>
<td>34.46</td>
<td>65.35</td>
<td>34.28</td>
<td>46.45</td>
</tr>
<tr>
<td></td>
<td>8634</td>
<td>8634</td>
<td>11274</td>
<td>7174</td>
</tr>
</tbody>
</table>

Table 1: Classification accuracy when replacing a single hidden layer with structured classes. Where applicable, rank \((r)\) is in parentheses, and the number of parameters in the architecture is in italics below each method. Comparisons to previously unexplored classic LDR types as well as additional baselines are included, with the ranks (or number of hidden units, as in the NN baseline which simply reduces the number of hidden units in the layer [7]) adjusted to match the parameter count of LDR-TD where possible. As the Fastfood [44] and Circulant [8] methods do not have rank parameters, the parameter count for these methods cannot be exactly controlled. Additional results when replacing the FC layer of a CNN are in Appendix E Additional details for all experiments are in Appendix F.

3In addition to the results reported in Table 1 Figure 4 and Table 7 in Appendix E we also found that at rank 16 the LDR-SD class on the CNN architecture achieve test accuracies of 68.48% and 75.45% on CIFAR-10 and NORB respectively.
At rank one (the most compressed setting), our classes with learned operators achieve higher accuracy than the fixed operator classes, and on the MNIST-bg-rot, MNIST-noise, and NORB datasets even improve on FC layers of the same dimensions, by 1.73, 13.30, and 2.92 accuracy points respectively on the SHL task, as shown in Table 1. On the CNN task, our classes improve upon fully-connected layers by 0.85 and 2.25 accuracy points on the MNIST-bg-rot and MNIST-noise datasets (shown in Table 7 in Appendix E). As noted above, at higher ranks our classes improve upon FC layers on all datasets in both the SHL and CNN architectures.

Additionally, in Figure 4 we compare the performance of LDR-SD at higher ranks. Note that the ratio of parameters between LDR-SD and Toeplitz-like or low-rank is \( r + 1 \), which becomes negligible at higher ranks. Figure 4 shows that at just rank 16, the LDR-SD class exceeds the performance of the FC layer on all four datasets, by 5.87, 15.05, 0.74, and 6.86 accuracy points on MNIST-bg-rot, MNIST-noise, CIFAR-10, and NORB respectively, while still maintaining at least 20X fewer parameters.

Of particular note is the poor performance of low-rank matrices. As mentioned in Section 2, every fixed-operator class has the same parametrization (a low-rank matrix). We hypothesize that the main contribution to their marked performance difference is the effect of the learned displacement operator modeling latent invariances in the data, and that the improvement in the displacement rank classes – from low-rank to Toeplitz-like to our learned operators – comes from more accurate representations of these invariances. As shown in Figure 4, broadening the operator class (from Toeplitz-like at \( r = 1 \) to LDR-SD at \( r = 1 \)) is consistently a more effective use of parameters than increasing the displacement rank (from Toeplitz-like at \( r = 1 \) to \( r = 2 \)). Note that LDR-SD \( (r = 1) \) and Toeplitz-like \( (r = 2) \) have the same parameter count.

For the rest of our experiments outside Section 5.1 we implemented the algorithms in Appendix C specifically for LDR-SD matrices, and focus on further evaluation of this class on more expensive models.

Language modeling  Here, we replace the input-hidden weights in a single layer long short-term memory network (LSTM) for a language modeling task. We evaluate on the WikiText-2 dataset, consisting of 2M training tokens and a vocabulary size of 33K [31]. We compare to Toeplitz-like and low-rank baselines, both previously investigated for compressing recurrent nets [29]. As shown in Table 2, the learned operators improve upon the baselines for each budget tested (corresponding to ranks 1,2,4,8,16,24 for the learned class). Though our class does not outperform the FC layer, we did find that it achieves a significantly lower perplexity than the fixed Toeplitz-like class (by 19.94-42.92 perplexity points), suggesting that learning the displacement operator can help adapt to different domains.
<table>
<thead>
<tr>
<th>Num. Parameters</th>
<th>LDR-SD</th>
<th>Toeplitz-like</th>
<th>Low-rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048</td>
<td>166.97</td>
<td>186.91</td>
<td>205.72</td>
</tr>
<tr>
<td>3072</td>
<td>154.51</td>
<td>177.60</td>
<td>179.46</td>
</tr>
<tr>
<td>5120</td>
<td>141.91</td>
<td>178.07</td>
<td>172.38</td>
</tr>
<tr>
<td>9216</td>
<td>143.60</td>
<td>186.52</td>
<td>144.41</td>
</tr>
<tr>
<td>17408</td>
<td>132.43</td>
<td>162.58</td>
<td>135.65</td>
</tr>
<tr>
<td>25600</td>
<td>129.46</td>
<td>155.73</td>
<td>133.37</td>
</tr>
</tbody>
</table>

Table 2: Test perplexity when replacing input-hidden matrices of an LSTM with structured classes on WikiText-2. A fully-connected weight layer, with 65536 parameters, has perplexity 117.74. Lower is better.

5.2 Replacing convolutional layers

Convolutional layers of CNNs are a prominent example of equivariant feature maps. However, it has been noted that convolutions are a subcase of Toeplitz-like matrices [8, 40], and the block closure property implies that multi-channel convolution filters are just a Toeplitz-like matrix of higher rank (see Appendix C, Corollary 1). In light of the possible approximate-equivariance interpretation of LDR, we are interested in whether replacing convolution layers with more general representations can recover similar performance, without needing the hand-crafted sparsity pattern.

Briefly, we test the simplest multi-channel CNN model on the CIFAR-10 dataset, consisting of one layer of convolution channels (3 in channels, 3 out channels), followed by a fully-connected layer, followed by the softmax layer. The final accuracies are listed in Table 3. The most striking result is for the simple architecture consisting of two layers of a single structured matrix. This comes within 1.1 accuracy points of the highly specialized architecture consisting of convolution channels + pooling + FC layer, while using fewer layers, hidden units, and parameters. The full details are in Appendix F.

Table 3: Replacing a 5 layer CNN consisting of convolution channels, max pooling, and FC layers with two generic LDR matrices results in only slight accuracy decrease while containing fewer layers, hidden units, and parameters. Rank (r) is in parentheses.

5.3 Generalization and sample complexity

Theorem 2 states that the theoretical sample complexity of neural networks with structured weight matrices scales almost linearly in the total number of parameters, matching the results for networks with fully-connected layers [4, 5]. As LDR matrices have much fewer parameters, the VC dimension bound for LDR networks are much lower than that of fully-connected networks. Though the VC dimension bounds are sufficient but not necessary for learnability, one might still expect to be able to learn over compressed networks with fewer samples than over fully-connected networks. We empirically investigate this result by testing whether structured classes have lower generalization error than a fully-connected layer on the single hidden layer model, and show in Table 9 (Appendix E) that structured matrices consistently have lower generalization error than the fully-connected baseline on all four datasets.

4Convolutions are designed to be shift equivariant, i.e. shifting the input is equivalent to shifting the output.
5We use the standard measure of the difference between training and test error.
Reducing sample complexity  We investigate whether LDR models with learned displacement operators require fewer samples to achieve the same test error, compared to fully-connected weights, in both a fully-connected and convolutional neural network. Tables 10 and 11 in Appendix F show our results. On the SHL model, when using only 25% of the training data the tridiagonal class exceeds the performance of a fully-connected layer trained on the full MNIST-noise dataset. On the CNN model, only 50% of the training data is sufficient for the tridiagonal class to exceed the performance of a fully-connected layer trained on the full dataset.

5.4 Visualizing learned weights

Finally, we examine the actual structures that our models learn. Figure 5(a,b) shows the heat map of the weight matrix $W \in \mathbb{R}^{784 \times 784}$ for the Toeplitz-like and LDR-SD classes, trained on MNIST-bg-rot with a single hidden layer model. As is convention, the input is flattened to a vector in $\mathbb{R}^{784}$ and the model does not know about the implicit 2D structure. The Toeplitz-like class is unable to determine that the input is actually a $28 \times 28$ image instead of a vector. In contrast, the matrix with the learned subdiagonal operator is able to pick up regularity in the input, as the weight matrix displays grid-like periodicity of size 28.

Figure 5(c) reveals why the weight matrix displays this pattern. The equivariance interpretation (Section 4) predicts that $B$ should encode a meaningful transformation of the inputs. The entries of the learned subdiagonal are in fact recovering a latent invariant of the 2D domain: when visualized as an image, it corresponds to how the inputs are centered (Figure 5(d)). Figure 6 in Appendix E shows a similar figure for the NORB dataset, which has smaller objects, and the subdiagonal learns a correspondingly smaller circle.

6 Conclusion

We substantially generalize the class of low displacement rank matrices explored in machine learning by considering classes of LDR matrices with displacement operators that can be learned from data. We show these matrices can improve performance on downstream tasks compared to compression baselines and even fully-connected weight layers, while retaining comparable compression factors to previous approaches. We hope this work inspires additional ways of using structure to achieve both more compact and higher quality representations, especially for deep learning models which are commonly acknowledged to be overparametrized.

Acknowledgments

We thank Taco Cohen, Braden Hancock, Tatsunori Hashimoto, Fred Sala, Virginia Smith, James Thomas, Mary Wootters, Paroma Varma, and Jian Zhang for helpful discussions and feedback.
This material is based on research sponsored by Defense Advanced Research Projects Agency (DARPA) under agreement number FA8750-17-2-0095. We gratefully acknowledge the support of the DARPA SIMPLEX program under No. N66001-15-C-4043, DARPA FA8750-12-2-0335 and FA8750-13-2-0039, DOE 108845, National Institute of Health (NIH) U54EB020405, the National Science Foundation (NSF) under awards No. CCF-1563078 and No. CCF-1763481, the Office of Naval Research (ONR) under awards No. N00014-17-1-2266, the Moore Foundation, the Okawa Research Grant, American Family Insurance, Accenture, Toshiba, and Intel. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project: Intel, Microsoft, Teradata, and VMware. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, AFRL, NSF, NIH, ONR, or the U.S. Government.

References


A Symbols and abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Used For</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR</td>
<td>low displacement rank</td>
</tr>
<tr>
<td>LDR-SD</td>
<td>matrices with low displacement rank with respect to subdiagonal operators</td>
</tr>
<tr>
<td>LDR-TD</td>
<td>matrices with low displacement rank with respect to tridiagonal operators</td>
</tr>
<tr>
<td>(A, B)</td>
<td>displacement operators</td>
</tr>
<tr>
<td>$\nabla_{A,B}[M]$</td>
<td>Sylvester displacement, $AM - MB$</td>
</tr>
<tr>
<td>(G, H)</td>
<td>factorization of $\nabla_{A,B}$, i.e. $\nabla_{A,B} = GH^T$.</td>
</tr>
<tr>
<td>$Z_f$</td>
<td>unit-f-circulant matrix.</td>
</tr>
<tr>
<td>$K(A, v)$</td>
<td>Krylov matrix, with $i$th column $A^i v$.</td>
</tr>
<tr>
<td>$\mathcal{D}^r_{A,B}$</td>
<td>matrices of displacement rank $\leq r$ with respect to $(A, B)$</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>feature map</td>
</tr>
<tr>
<td>$L$</td>
<td>number of layers</td>
</tr>
<tr>
<td>$r$</td>
<td>(displacement) rank</td>
</tr>
<tr>
<td>CC</td>
<td>convolutional channels</td>
</tr>
<tr>
<td>FC</td>
<td>fully-connected</td>
</tr>
</tbody>
</table>

Table 4: Symbols and abbreviations used in this paper.

B Related work

Our study of the potential for structured matrices for compressing deep learning pipelines was motivated by exciting work along these lines from Sindhwani et al. [40], the first to suggest the use of low displacement rank (LDR) matrices in deep learning. They specifically explored applications of the Toeplitz-like class, inspired by the fact that these generalize one-dimensional convolutions, and empirically show that they are competitive against many other baselines for compressing CNNs on image and speech domains. Toeplitz-like matrices were similarly found to be effective at compressing RNN and LSTM architectures on a voice search task [30]. Another special case of LDR matrices are the circulant (or block-circulant) matrices, which have also been used for compressing CNNs [8]; more recently, these have also been tackled from an architectural and hardware angle [17]. Earlier works on compressing deep learning pipelines analyzed low-rank matrices [15, 37] – perhaps the most canonical type of dense structured matrix – which are also encompassed by our framework. Outside of neural networks, Choromanski et al. examined a structured matrix class that includes Toeplitz-like, circulant, and Hankel matrices (which are all LDR matrices) in the context of kernel approximation [10].

On the theoretical side, Zhao et al. [45] study properties of neural networks with LDR weight matrices, proving results including a universal approximation property and error bounds. However, they retain the standard paradigm of fixing the displacement operators and varying the low-rank portion. Another natural theoretical question that arises with these models is whether the resulting hypothesis class is still efficiently learnable, especially when learning the structured class (as opposed to these previous fixed classes). Theorem 2 provides the first sample complexity bounds (to the best of our knowledge) for neural networks with a broad class of structured weight matrices including low-rank, our LDR classes, and other general structured matrices [14]. Additionally, a recent preprint proved a Rademacher complexity bound for one layer neural networks with low-rank weight matrices [23].

In Section 3 we suggest that the LDR representation enforces approximate equivariance and satisfies natural closure properties that one would expect of equivariant representations. The study of equivariant maps is of broad interest for constructing more effective representations when known symmetries exist in underlying data. The fact that convolutional networks induce equivariant representations, and the importance of this effect on sample complexity and generalization, has been well-analyzed [2, 19, 41]. Building upon the observation that convolutional filters are simply linear maps constructed to be translation equivariant,

\footnote{Shifting the input to a convolution feature map is the same as shifting the output.}
exciting recent progress has been made on crafting representations invariant to more complicated symmetries such as the spherical rotation group \[13\] and egomotions \[1\].

Generally, however, underlying assumptions need to be made about the domain and invariances present in order to construct feature maps for each application.

A few works have explored the possibility of learning invariances automatically from data, and design deep architectures that are in principle capable of modeling and learning more general symmetries \[18, 34\].

C Properties of displacement rank

Displacement rank has traditionally been used to describe the Toeplitz-like, Hankel-like, Vandermonde-like, and Cauchy-like matrices, which are ubiquitous in disciplines such as engineering, coding theory, and computer algebra. Their associated displacement representations are shown in Table 5.

<table>
<thead>
<tr>
<th>Structured Matrix</th>
<th>A</th>
<th>B</th>
<th>Displacement Rank r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toeplitz</td>
<td>(Z_1)</td>
<td>(Z_{-1})</td>
<td>(\leq 2)</td>
</tr>
<tr>
<td>Hankel</td>
<td>(Z_1)</td>
<td>(Z_0)</td>
<td>(\leq 2)</td>
</tr>
<tr>
<td>Vandermonde</td>
<td>diag((r))</td>
<td>(Z_0)</td>
<td>(\leq 1)</td>
</tr>
<tr>
<td>Cauchy</td>
<td>diag((s))</td>
<td>diag((t))</td>
<td>(\leq 1)</td>
</tr>
</tbody>
</table>

Table 5: Traditional classes of structured matrices analyzed with displacement rank.

Proof of Proposition \[7\] The following identities are easily verified:

**Transpose**

\[
\nabla_{B^\tau, A^\tau} M^T = - (\nabla_{A, B} M)^T
\]

**Inverse**

\[
\nabla_{B, A} M^{-1} = -M^{-1} (\nabla_{A, B} M) M^{-1}
\]

**Sum**

\[
\nabla_{A, B} (M + N) = \nabla_{A, B} M + \nabla_{A, B} N
\]

**Product**

\[
\nabla_{A, C} M N = (\nabla_{A, B} M) N + M (\nabla_{B, C} N)
\]

**Block** The remainder

\[
\text{diag}(A_1, \ldots, A_k) M - M \text{diag}(B_1, \ldots, B_\ell)
\]

is the block matrix

\[
(\nabla_{A_i, B_j} M_{ij})_{1 \leq i \leq k, 1 \leq j \leq \ell}.
\]

This is the sum of \(k\ell\) matrices of rank \(r\) and thus has rank \(rk\ell\).

**Corollary 1.** A \(k \times \ell\) block matrix \(M\), where each block is a Toeplitz-like matrix of displacement rank \(r\), is Toeplitz-like with displacement rank \(rk\ell + 2k + 2\ell\).

**Proof.** Apply Proposition \[d\] where each \(A_k, B_k\) has the form \(Z_f\). Let \(A = \text{diag}(A_1, \ldots, A_k)\) and \(B = \text{diag}(B_1, \ldots, B_\ell)\). Note that \(A\) and \(Z_1\) (of the same size as \(A\)) differ only in \(2k\) entries, and similarly \(B\) and \(Z_{-1}\) differ in \(2\ell\) entries. Since an \(s\)-sparse matrix also has rank at most \(s\),

\[
Z_1 M - M Z_{-1} = AM - MB + (Z_1 - A) M - M(Z_{-1} - B)
\]

has rank at most \(rk\ell + 2k + 2\ell\).
For example, \[40\] defines the Toeplitz-like matrices as
which is
a
where
The rank
will only be non-empty in the first column, hence
the Discrete Fourier Transform (DFT) and Discrete Cosine Transform (DCT) for their ubiquity.
Trigonometric transforms
Orthogonal polynomial transforms include many special cases. We single out
Every orthogonal polynomial family satisfies a three-term recurrence
Proposition 4. orthogonal polynomial transforms have displacement rank 1 with respect to tridiagonal operators.
Proof. Every orthogonal polynomial family satisfies a three-term recurrence
\[ p_{i+1}(X) = (a_iX + b_i)p_i(X) + c_ip_{i-1}(X) \] (4)
where \(a_i > 0\). Let \(M\) be an orthogonal polynomial transform with respect to polynomials \((p_i(X))_{0 \leq i < n}\)
and nodes \((\lambda_j)_{0 \leq j < n}\). Define the tridiagonal and diagonal matrix
\[
A = \begin{bmatrix}
\frac{b_0}{\alpha_0} & \frac{1}{\alpha_0} & 0 & \ldots & 0 & 0 \\
-\frac{c_1}{\alpha_1} & -\frac{b_0}{\alpha_1} & \frac{1}{\alpha_1} & \ldots & 0 & 0 \\
0 & -\frac{c_1}{\alpha_1} & -\frac{b_0}{\alpha_1} & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & \frac{b_{m-2}}{\alpha_{m-2}} & \frac{1}{\alpha_{m-1}} \\
0 & 0 & 0 & \ldots & \frac{c_{m-2}}{\alpha_{m-1}} & -\frac{b_{m-1}}{\alpha_{m-1}} \\
\end{bmatrix}
\]
\[
B = \text{diag}(\lambda_0, \ldots, \lambda_{n-1}).
\]
For any \(i \in \{0, \ldots, m - 2\}\) and any \(j\), consider entry \(ij\) of \(AM - MB\). This is
\[
\frac{1}{a_i} \left[ -c_ip_{i-1}(\lambda_j) - b_ip_i(\lambda_j) + p_{i+1}(\lambda_j) - \lambda_jp_i(\lambda_j) \right]
\]
which is 0 by plugging \(\lambda_j\) into (4).
Thus \(\nabla_{A,B}M\) can only non-zero in the last row, so \(M \in D^1_{A,B}\).

**Trigonometric transforms**
Orthogonal polynomial transforms include many special cases. We single out
the Discrete Fourier Transform (DFT) and Discrete Cosine Transform (DCT) for their ubiquity.
The \(N \times N\) DFT and DCT (type II) are defined as matrix multiplication by the matrices
\[
F = \left(e^{-2\pi i \frac{\pi}{N}} \right)_{ij}
\]
\[
C = \left(\cos \left[\frac{\pi}{N} i(j + 1/2)\right] \right)_{ij}
\]
respectively.

The former is a special type of Vandermonde matrix, which were already shown to be in $D_{td}$. Also note that Vandermonde matrices $(\lambda^j_i)_{ij}$ are themselves orthogonal polynomial transforms with $p_i(X) = X^i$.

The latter can be written as

$$
\left( T_i \left( \cos \left[ \frac{\pi}{N} (j + \frac{1}{2}) \right] \right) \right)_{ij},
$$

where $T_i$ are the Chebyshev polynomials (of the first kind) defined such that

$$
T_n(X) = \cos(n \arccos x).
$$

Thus this is an orthogonal polynomial transform with respect to the Chebyshev polynomials.

Other constructions From these basic building blocks, interesting constructions belonging to $D_{td}$ can be found via the closure properties. For example, several types of structured layers inspired by convolutions, including Toeplitz [10], circulant [8] and block-circulant [17] matrices, are special instances of Toeplitz-like matrices. We also point out a more sophisticated layer [32] in the tridiagonal LDR class, which requires more deliberate use of Proposition 1 to show.

Proposition 5. The ACDC$^{-1}$ layer, where $A, D$ are diagonal matrices and $C$ is the Discrete Cosine Transform [32], has displacement rank 2 with respect to tridiagonal operators.

Proof. Let $T, \Lambda$ be the tridiagonal and diagonal matrix such that $C \in D_{T, \Lambda}$. Define $S = ATA^{-1}$, which is also tridiagonal. Note that $A \in D_{S, T}$ by construction. Also note that $D \in D_{\Lambda, \Lambda}$ since $\Lambda$ is diagonal. An application of the inverse closure rule yields $C \in D_{A, T}$. Finally, the product closure property implies that

$$
ACDC^{-1} \in D_{S, T}.
$$

C.2 Algorithm derivation and details

De Sa et al. recently showed that a very general class of LDR matrices have asymptotically fast matrix-vector multiplication algorithms [14]. However, parts of the argument are left to existential results. Building off De Sa et al. [14], we derive a simplified and self-contained algorithm for multiplication by LDR matrices with subdiagonal operators.

Since these matrices can be represented by the Krylov product formula (2), it suffices to show multiplication algorithms separately for matrix-vector multiplication by $K(A, v)^T$ and $K(A, v)$.

Krylov transpose multiplication Let $A \in \mathbb{R}^{n \times n}$ be a subdiagonal matrix, i.e. $A_{i+1,i}$ are the only possible non-zero entries. Let $u, v \in \mathbb{R}^n$, we wish to compute the product $K(A, v)^Tu$. For simplicity assume $n$ is a power of 2.

Following [14], the vector

$$
u^TK(A, v) = [uv \ uAv \ \ldots \ uA^{n-1}v]\n$$

is the coefficient vector of the polynomial in $X$

$$
uv + uAvX + \cdots + uA^{n-1}vX^{n-1}
= \sum_{i=0}^{\infty} uA^iX^iv
= u(I - AX)^{-1}v,
$$

where we use the observation that $A^n = 0$. 

17
By partitioning \( A \) into \( n/2 \times n/2 \) blocks, it has the form \[
\begin{bmatrix}
A_0 & 0 \\
e_n^T e_n & A_1
\end{bmatrix},
\] where \( A_0, A_1 \) are subdiagonal matrices of half the size, \( a \) is a scalar, and \( e_i \) are basis vectors. Let also \( u_0, u_1 \in \mathbb{R}^{n/2} \), \( v_0, v_1 \in \mathbb{R}^{n/2} \) denote the first and second halves of \( u, v \).

By block matrix inversion for triangular matrices \[
\begin{bmatrix} A & 0 \\ C & B \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ -B^{-1}CA^{-1} & B^{-1} \end{bmatrix},
\] this can be written as

\[
u^T (I - AX)^{-1} v = \left[ u_0^T \quad u_1^T \right] \begin{bmatrix}
(I - A_0 X)^{-1} & 0 \\
-(I - A_1 X)^{-1} (-ae_1 e_n^T Y) (I - A_0 X)^{-1} & (I - A_1 X)^{-1}
\end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}
\]

\[
u_0^T (I - A_0 X)^{-1} v_0 + u_1^T (I - A_1 X)^{-1} v_1 + a X (u_1^T (I - A_1 X)^{-1} e_1) \left( e_n^T (I - A_0 X)^{-1} v_0 \right)
\]

Therefore \( u^T (I - AX)^{-1} v \) can be computed from

\[
\begin{align*}
u_0^T (I - A_0 X)^{-1} v_0 & \quad u_1^T (I - A_1 X)^{-1} v_1 \\
u_1^T (I - A_1 X)^{-1} e_1 & \quad e_n^T (I - A_0 X)^{-1} v_0
\end{align*}
\]

with an additional polynomial multiplication and 3 polynomial addition/subtractions.

A modification of this reduction shows that the \( 2 \times 2 \) matrix of polynomials \[
[u \quad e_n]^T (I - AX)^{-1} [v \quad e_1]
\]

can be computed from

\[
\begin{bmatrix} u_0 & e_n \end{bmatrix}^T (I - A_0 X)^{-1} [v_0 \quad e_1] \quad [u_1 & e_n]^T (I - A_1 X)^{-1} [v_1 \quad e_1]
\]

with an additional constant number of polynomial multiplications and additions.

The complete recursive algorithm is provided in Algorithm 1 where subroutine \( \mathbf{R} \) computes the above matrix of polynomials. For convenience, Algorithm 1 uses Python indexing notation.

\begin{algorithm}
\caption{Krylov Transpose (Recursive)}
\begin{algorithmic}[1]
\Function{KrylovTranspose}{\( A \in \mathbb{R}^{n \times n}, u, v \in \mathbb{R}^n \)}
\State \( s \leftarrow \text{subdiagonal}(A) \)
\State \Return \( \mathbf{R}(s, u, v) \)
\EndFunction
\Function{R}{\( s \in \mathbb{R}^{n-1}, u, v \)}
\State \( S_0 \leftarrow \mathbf{R}(s[0 : n/2 - 1], u[0 : n/2], v[0 : n/2]) \)
\State \( S_1 \leftarrow \mathbf{R}(s[n/2 : n - 1], u[n/2 : n], v[n/2 : n]) \)
\State \( \mathcal{L} \leftarrow s[n/2 - 1] \cdot \begin{bmatrix} S_1[0, 1] \cdot S_0[0, 0] \\
S_1[1, 1] \cdot S_0[0, 0] \end{bmatrix} \quad S_1[0, 1] \cdot S_0[1, 1] \quad S_1[1, 1] \cdot S_0[1, 1] \)
\State \Return \[
\begin{bmatrix}
L[0, 0] + S_0[0, 0] + S_1[0, 0] & L[0, 1] + S_0[0, 1] \\
L[1, 0] + S_1[1, 0] & L[1, 1]
\end{bmatrix}
\]
\EndFunction
\end{algorithmic}
\end{algorithm}

A polynomial multiplication of degree \( m \) in Step 8 can be computed as a convolution of size \( 2m \). This reduces to two Fast Fourier Transform (FFT) calls, an elementwise multiplication in the frequency domain, and an inverse FFT. The total number of calls can be further reduced to \( 4 \) FFTs and \( 4 \) inverse FFTs.

Algorithm 1 defines a recursion tree, and in practice we compute this breadth first bottom-up to avoid recursive overhead. This also allows the FFT operations to be batched and computed in parallel. Thus the \( d \)-th layer of the algorithm (starting from the leaves) performs \( \frac{n}{2^d} \) FFT computations of size \( 2^{d+1} \).

This completes the proof of Theorem 1.

\textbf{Krylov multiplication} [4] do not provide explicit algorithms for the more complicated problem of multiplication by \( K(A, v) \), instead justifying the existence of such an algorithm with the \textbf{transposition principle}. Traditional proofs of the transposition principle use circuit based arguments involving reversing arrows in the arithmetic circuit defining the algorithm’s computation graph [6].
Here we show an alternative simple way to implement the transpose algorithm using any automatic differentiation (AD) implementation, which all modern deep learning frameworks include. AD states that for any computation, its derivative can be computed with only a constant factor more operations [20].

**Proposition 6** (Transposition Principle). If the matrix \( M \in \mathbb{R}^{n \times n} \) admits matrix-vector multiplication by any vector in \( N \) operations, then \( M^T \) admits matrix-vector multiplication in \( O(N + n) \) operations.

**Proof.** Note that for any \( x \) and \( y \), the scalar \( y^T M x = y \cdot (M x) \) can be computed in \( N + n \) operations.

The statement follows from applying reverse-mode AD to compute \( \frac{\partial}{\partial x} y^T M x \).

Additionally, the algorithm can be optimized by choosing \( x = 0 \) to construct the forward graph. \( \Box \)

We have released PyTorch implementations for all algorithms mentioned in this section. To perform the optimization mentioned in Proposition [6] and avoid needing second-order derivatives when computing backprop for gradient descent, we provide an explicit implementation of non-transpose Krylov multiplication \( K(A, v) \). This was found by using Proposition [6] to hand-differentiate Algorithm [1].

Finally, we comment on multiplication by the LDR-TD class. Desa et al.[14] showed that these matrices also have asymptotically efficient multiplication algorithms, of the order \( O(r n \log^3 n) \) operations. However, these algorithms are even more complicated and involve operations such as inverting matrices of polynomials in a modulus. Practical algorithms for this class similar to the one we provide for LDR-SD matrices require more work to derive.

### C.3 Displacement rank and equivariance

Here we discuss in more detail the connection between LDR and equivariance. One line of work [12,24] has used the group representation theory formalization of equivariant maps, in which the model is equivariant to a set of transformations which form a group \( G \). Each transformation \( g \in G \) acts on an input \( x \) via a corresponding linear map \( T_g \). For example, elements of the rotation group in two and three dimensions, \( SO(2) \) and \( SO(3) \), can be represented by 2D and 3D rotation matrices respectively. Formally, a feature map \( \Phi \) is equivariant if it satisfies

\[
\Phi(T_g x) = T'_{g'} \Phi(x)
\]

for representations \( T, T' \) of \( G \) [12,24]. This means that perturbing the input \( x \) by a transformation \( g \in G \) before computing the map \( \Phi \) is equivalent to first finding the features \( \Phi \) and then applying the transformation.

Group equivariant convolutional neural networks (G-CNNs) are a particular realization where \( \Phi \) has a specific form \( G \to \mathbb{R}^d \), and \( T, T' \) are chosen in advance [12]. We use the notation \( \Phi \) to distinguish our setting, where the input \( x \) is finite dimensional and \( \Phi \) is linear.

**Proposition 7.** If \( \Phi \) has displacement rank 0 with respect to invertible \( A, B \), then \( \Phi \) is equivariant as defined by [7].

**Proof.** Note that if \( A \Phi = \Phi B \) for invertible matrices \( A, B \) (i.e. if a matrix \( \Phi \) has displacement rank 0 with respect to \( A \) and \( B \)), then \( A^i \Phi = \Phi B^i \) also holds for \( i \in \mathbb{Z} \). Also note that the set of powers of any invertible matrix forms a cyclic group, where the group operation is multiplication. The statement follows directly from this fact, where the group \( G \) is \( \mathbb{Z} \), and the representations \( T \) and \( T' \) of \( G \) correspond to the cyclic groups generated by \( A \) and \( B \), respectively consisting of \( A^i \) and \( B^i \) for all \( i \in \mathbb{Z} \).

More generally, a feature map \( \Phi \) satisfying [7] for a set of generators \( S = \{g_i\} \) is equivariant with respect to the free group generated by \( S \). Proposition [7] follows from the specific case of a single generator, i.e. \( S = \{1\} \).

### D Bound on VC dimension and sample complexity

In this section we upper bound the VC dimension of a neural network where all the weight matrices are LDR matrices and the activation functions are piecewise polynomials. In particular, the VC dimension is almost linear in the number of parameters, which is much smaller than the VC dimension of a network with fully-connected layers. The bound on the VC dimension allows us to bound the sample complexity to learn an LDR network that performs well among LDR networks. This formalizes the intuition that compressed parameterization reduces the complexity of the class.
Neural network model  Consider a neural network architecture with $W$ parameters, arranged in $L$ layers. Each layer $l$, has output dimension $n_l$, where $n_0$ is the dimension of the input data and the output dimension is $n_L = 1$. For $l = 1, \ldots, L$, let $i_l \in \mathbb{R}^{n_l}$ be the input to the $l$-th layer. The input to the $(l+1)$-th layer is exactly the output of the $l$-th layer. The activation functions $\phi_l$ are piecewise polynomials with at most $p+1$ pieces and degree at most $k \geq 1$. The input to the first layer is the data $i_1 = x \in \mathbb{R}^{n_1}$, and the output of the last layer is a real number $i_{L+1} \in \mathbb{R}$. The intermediate layer computation has the form:

$$i_{l+1} = \phi_l(M_i i_l + b_l) \text{ (applied elementwise),} \quad \text{where } M_l \in \mathbb{R}^{n_{l-1} \times n_l}, \ b_l \in \mathbb{R}^{n_l}.$$

We assume the activation function of the final layer is the identity.

Each weight matrix $M_l$ is defined through some set of parameters; for example, traditional unconstrained matrices are parametrized by their entries, and our formulation \((2)\) is parametrized by the entries of some operator matrices $A_l, B_l$ and low-rank matrix $G_l H_l^T$. We collectively refer to them all the parameters of the neural network (including the biases $b_l$) as $\theta \in \mathbb{R}^W$, where $W$ is the number of parameters.

Bounding the polynomial degree  The crux of the proof of the VC dimension bound is that the entries of $M \in \mathbb{R}^{n \times m}$ are polynomials in terms of the entries of its parameters $(A, B, G, H)$, of total degree at most $c_1 m^{c_2}$ for universal constants $c_1, c_2$. This allows us to bound the total degree of all of the layers and apply Warren’s lemma to bound the VC dimension.

We will first show this for the specific class of matrices that we use, where the matrix $M$ is defined through equation \((2)\).

**Lemma 1.** Suppose that $M \in \mathbb{R}^{n \times m}$ is defined as

$$M = \sum_{i=1}^r K(A, g_i) K(B^T, h_i).$$

Then the entries of $M$ are polynomials of the entries of $A, B, G, H$ with total degree at most $2m$.

**Proof.** Since $K(A, g_i) = [g_i, A g_i, \ldots, A^{m-1} g_i]$, and each entry of $A^k$ is a polynomial of the entries of $A$ with total degree at most $k$, the entries of $K(A, g_i)$ are polynomials of the entries of $A$ and $g_i$ with total degree at most $m$. Similarly the entries of $K(B^T, h_i)$ are polynomials of the entries of $B$ and $h_i$ with total degree at most $m$. Hence the entries of $K(A, g_i) K(B^T, h_i)$ are polynomials of the entries of $A, B, G, H$ with total degree at most $2m$. We then conclude that the entries of $M$ are polynomials of the entries of $A, B, G, H$ with total degree at most $2m$. \(\square\)

**Lemma 2.** For a fixed data point $x$, at the $l$-th layer of a neural network with LDR weight matrices, each entry of $M_l i_l + b_l$ is a piecewise polynomial of the network parameters $\theta$, with total degree at most $d_l$, where

$$d_0 = 0, \quad d_l = k d_{l-1} + c_1 n_{l-1}^{c_2} \quad \text{for } \ l = 1, \ldots, L,$$

where $c_1$ and $c_2$ are universal constants defined in Lemma 1. Thus entries of the output $\phi_l(M_l i_l + b_l)$ are piecewise polynomials of $\theta$ with total degree at most $kd_l$. Moreover,

$$d_l \leq c_1 k^{l-1} \sum_{j=0}^{l-1} n_j^{c_2}. \quad (6)$$

**Proof.** We induct on $l$. For $l = 1$, since $i_1 = x$ is fixed, the entries of $M_1$ are polynomials of $\theta$ of degree at most $c_1 n_0^{c_2}$ by Lemma 1 and so the entries of $M_1 i_1 + b_1$ are polynomials of $\theta$ with total degree at most $d_1 = c_1 n_0^{c_2}$. As $\phi$ is a piecewise polynomials of degree at most $k$, each entry the output $\phi_1(M_1 i_1 + b_1)$ is a piecewise polynomial of $\theta$ with total degree at most $2n_0 k$. The bound \((6)\) holds trivially.

Suppose that the lemma is true for some $l - 1 > 1$. Since the entries of $i_l$ are piecewise polynomials of $\theta$ with total degree at most $k d_{l-1}$ and entries of $M_l$ are polynomials of $\theta$ with total degree at most $c_1 n_{l-1}^{c_2}$ by Lemma 1 the entries of $M_l i_l + b_l$ are piecewise polynomials of $\theta$ with total degree at most $d_l = k d_{l-1} + c_1 n_{l-1}^{c_2}$. Thus $\phi_l(M_l i_l + b_l)$ have entries that are piecewise polynomials of $\theta$ with total degree at most $k d_l$.\(\square\)
We can bound

\[ d_l = kd_{l-1} + c_1n_{l-1}^2 \leq kc_1k^{l-2}n_{l-1}^2 + c_1k^{l-1}n_{l-2}^2, \]

where we have used the fact that \( k \geq 1 \), so \( c_1n_{l-1}^2 \leq c_1k^{l-1}n_{l-2}^2 \). This concludes the proof. \(\Box\)

**Bounding the VC dimension** Now we are ready to bound the VC dimension of the neural network.

**Theorem 3.** For input \( x \in \mathcal{X} \) and parameter \( \theta \in \mathbb{R}^W \), let \( f(x, \theta) \) denote the output of the network. Let \( \mathcal{F} \) be the class of functions \( \{ x \to f(x, \theta) : \theta \in \mathbb{R}^W \} \). Let \( W_l \) be the number of parameters up to layer \( l \) (i.e., the total number of parameters in layer 1, 2, \ldots, \( l \)). Define the effective depth as

\[ \bar{L} := \frac{1}{W} \sum_{l=1}^{L} W_l, \]

and the total number of computation units (including the input dimension) as

\[ U := \sum_{l=0}^{L} n_l. \]

Then

\[ \text{VCdim}(\text{sign } \mathcal{F}) = O(\bar{L}W \log(pU) + \bar{L}W \log k). \]

In particular, if \( k = 1 \) (corresponding to piecewise linear networks) then

\[ \text{VCdim}(\text{sign } \mathcal{F}) = O(\bar{L}W \log(pU)) = O(LW \log W). \]

We adapt the proof of the upper bound from Bartlett et al. [4, 5]. The main technical tool is Warren’s lemma [43], which bounds the growth function of a set of polynomials. We state a slightly improved form here from Anthony and Bartlett [3, Theorem 8.3].

**Lemma 3.** Let \( p_1, \ldots, p_m \) be polynomials of degree at most \( d \) in \( n \leq m \) variables. Define

\[ K := |\{(\text{sign } p_1(x)), \ldots, \text{sign } p_m(x) : x \in \mathbb{R}^n\}|, \]

i.e., \( K \) is the number of possible sign vectors given by the polynomials. Then \( K \leq 2(2emd/n)^n \).

**Proof of Theorem 3** Fixed some large integer \( m \) and some inputs \( x_1, \ldots, x_m \). We want to bound the number of sign patterns that the neural network can output for the set of input \( x_1, \ldots, x_m \):

\[ K := |\{(\text{sign } f(x_1, \theta), \ldots, \text{sign } f(x_m, \theta)) : \theta \in \mathbb{R}^W\}|. \]

We want to partition the parameter space \( \mathbb{R}^W \) so that for a fixed \( x_j \), the output \( f(x_j, \theta) \) is a polynomial on each region in the partition. Then we can apply Warren’s lemma to bound the number of sign patterns. Indeed, for any partition \( \mathcal{S} = \{ P_1, \ldots, P_N \} \) of the parameter space \( \mathbb{R}^W \), we have

\[ K \leq \sum_{j=1}^{N} |\{(\text{sign } f(x_1, \theta), \ldots, \text{sign } f(x_m, \theta)) : \theta \in P_j\}|. \]

We construct the partitions iteratively layer by layer, through a sequence \( \mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{L-1} \) of successive refinements, satisfying the two properties:
1. \(|S_0| = 1\) and for each \(1 \leq l \leq L - 1\),
\[
|S_l| \leq |S_{l-1}| \left( \frac{2empl_l}{W_l} \right)^{W_l},
\]
where \(n_l\) is the dimension of the output of the \(l\)-th layer, \(d_l\) is the bound on the total degree of \(M_l i_l + b_l\) as piecewise polynomials of \(\theta\) as defined in Lemma\(^2\) and \(W_l\) is the number of parameters up to layer \(l\) (i.e., the total number of parameters in layer \(1, 2, \ldots, l\)).

2. For each \(l = 0, \ldots, L - 1\), for each element \(S \in S_l\), for each fixed data point \(x_j\) (with \(j = 1, \ldots, m\)), the entries of the output \(\phi_i(M_l i_l + b_l)\) when restricted to \(S\) are polynomials of \(\theta\) with total degree at most \(kd_l\).

We can define \(S_0 = \mathbb{R}^W\), which satisfies property 2, since at layer 1, the entries of \(i_1 = x_j\) (for fixed \(x_j\)) are polynomials of \(\theta\) of degree \(d_0 = 0\).

Suppose that we have constructed \(S_0, \ldots, S_{l-1}\), and we want to define \(S_l\). For any \(h \in [n_l], j \in [m]\), and \(S \in S_{l-1}\), let \(p_{h, x_j, S}(\theta) = (M_l i_l + b_l)_{h\mid S}\) be the \(h\)-th entry of \(M_l i_l + b_l\) restricted to the region \(S\). By the induction hypothesis, for each \(S \in S_{l-1}\), the entries of \(i_l\) when restricted to \(S\) are polynomials of \(\theta\) of total degree at most \(kd_{l-1}\). Thus by Lemma\(^3\), the entries of \(M_l i_l + b_l\) when restricted to \(S\) are polynomials of \(\theta\) with total degree at most \(kd_{l-1} + c_1 n_l^{d_{l-1}} = d_l\), and depends on at most \(W_l\) many variables.

Let \(\{t_1, \ldots, t_p\}\) be the set of breakpoints of the activation function, which has at most \(p\) pieces. For any fixed \(S \in S_{l-1}\), by Lemma\(^4\) the collection of polynomials
\[
\{p_{h, x_j, S}(\theta) - t_i : h \in [n_l], j \in [m], i \in [p]\}
\]
attains at most
\[
\Pi := 2 \left( \frac{2e(n_mp)d_l}{W_l} \right)^{W_l}
\]
distinct sign patterns when \(\theta \in \mathbb{R}^W\). Thus, we can partition \(\mathbb{R}^W\) into this many regions, such that all these polynomials have the same signs within each region. We intersect all these regions with \(S\) to obtain a partition of \(S\) into at most \(\Pi\) subregions. Applying this for all \(S \in S_{l-1}\) gives our desired partition \(S_l\). Thus, the required property 1 is satisfied.

Fix some \(S' \in S_n\). Notice that, when \(\theta\) is restricted to \(S'\), all the polynomials
\[
\{p_{h, x_j, S}(\theta) - t_i : h \in [n_l], j \in [m], i \in [p]\}
\]
have the same sign, hence the entries of \(M_l i_l + b_l\) lies between two breakpoints of the activation function, and so the entries of the output \(\phi_i(M_l i_l + b_l)\) is a fixed polynomial in \(W_n\) variables of degree no more than \(kd_l\). This implies that the entries of the input \(i_{l+1}\) are polynomial function of \(W_l\) variables of degree no more than \(kd_l\).

This recursive construction yields a partition \(S_{L-1}\) of \(\mathbb{R}^W\) such that for \(S \in S_{L-1}\) the network output in response to any \(x_j\) is a fixed polynomial of \(\theta \in S\) of degree no more than \(kd_{L-1} + c_1 n_{L-1}^{d_{L-1}} = d_L\) (recall that we assume the activation function of the final layer is the identity). Hence by Lemma\(^3\) again,
\[
|\{(\sign f(x_1, \theta), \ldots, \sign f(x_m, \theta)) : \theta \in S\}| \leq 2 \left( \frac{2emkd_L}{W_L} \right)^{W_L}.
\]
By the property 1, we can bound the size of \(S_{L-1}\):
\[
|S_L| \leq \prod_{l=1}^{L-1} \left( \frac{2empl_l}{W_l} \right)^{W_l}.
\]
Combining the two bounds along with equation (7) yields
\[
K \leq \prod_{l=1}^{L} \left( \frac{2empl_l}{W_l} \right)^{W_l}.
\]
We can take logarithm and apply Jensen’s inequality, with \( \bar{W} := \sum_{i=1}^{L} W_i \):

\[
\log_2 K \leq L + \sum_{i=1}^{L} W_i \log_2 \frac{2emp_i d_i}{W_i} \\
= L + \bar{W} \sum_{i=1}^{L} \frac{W_i}{\bar{W}} \log_2 \frac{2emp_i d_i}{W_i} \\
\leq L + \bar{W} \log_2 \left( \sum_{i=1}^{L} \frac{W_i}{\bar{W}} \frac{2emp_i d_i}{W_i} \right) \quad \text{(Jensen’s inequality)} \\
= L + \bar{W} \log_2 \frac{2emp \sum_{i=1}^{L} n_i d_i}{W}.
\]

We can bound \( \sum n_i d_i \) using the bound on \( d_i \) from Lemma 2:

\[
\sum_{i=1}^{L} n_i d_i \leq \sum_{i=1}^{L} n_i c_1 k^{L-1} \sum_{j=0}^{L-1} n_j^{c_2} \leq LU c_1 k^{L-1} U^{c_2} \leq c_1 U^{c_2+2} k^{L},
\]

where we used the fact that \( L \leq U \). Thus

\[
\log_2 K \leq L + \bar{W} \log_2 \frac{2c_1 emp U^{2+c_2} k^{L}}{W}.
\]

To bound the VC-dimension, recall that by definition, if \( \text{VCdim}(\text{sign}\mathcal{F}) = m \) then exists \( m \) data points \( x_1, \ldots, x_m \) such that the output of the model can have \( 2^m \) sign patterns. The bound on \( \log_2 K \) then implies

\[
\text{VCdim}(\text{sign}\mathcal{F}) \leq L + \bar{W} \log_2 \frac{2c_1 emp U^{2+c_2} k^{L} \text{VCdim}(\text{sign}\mathcal{F})}{W}.
\]

We then use Lemma 4 below, noting that \( 2c_1 emp U^{2+c_2} k^{L} \geq 16 \), to conclude that

\[
\text{VCdim}(\text{sign}\mathcal{F}) \leq L + \bar{W} \log_2 (2c_1 emp U^{2+c_2} k^{L} \log_2 (2c_1 emp U^{2+c_2} k^{L})) = O(\bar{W} \log(pU) + \bar{W}L \log k),
\]

completing the proof.

A bound on the VC dimension immediate yields a bound on the sample complexity of learning from this class of neural networks with LDR matrices \([42]\).

**Corollary 2.** The class of neural network with LDR matrices as weights and piecewise linear activation is \((\epsilon, \delta)\)-PAC-learnable with a sample of size

\[
O \left( \frac{LW \log W + \log \frac{1}{\delta}}{\epsilon} \right).
\]

Since the number of parameters \( W \) is around the square root of the number of parameters of a network with fully-connected layers (assuming fixed rank of the LDR matrices), the sample complexity of LDR networks is much smaller than that of fully-connected networks.

**Lemma 4** (Lemma 16 of [5]). Suppose that \( 2^m \leq 2^r (mr/w)^w \) for some \( r \geq 16 \) and \( m \geq w \geq t \geq 0 \). Then, \( m \leq t + w \log_2 (2r \log_2 r) \).

**Extending to rational functions** We now show that Theorem 3 holds for matrices where the entries are rational functions – rather than polynomials – of its parameters, incurring only a constant in the bound. To define the function class \( \text{sign}\mathcal{F} \), we account for the possibility of poles by defining \( \text{sign}(a/0) = 0 \).

We only need to check that Lemma 2 and Lemma 3 still hold when polynomials are replaced by rational functions everywhere, and the degree of a rational function is defined as the usual \( \text{deg}(a/b) = \max\{\text{deg} a, \text{deg} b\} \).
To show Lemma 2 still holds, it suffices that the compositional degree bound \( \deg(f \circ g) \leq \deg(f) \deg(g) \) holds for rational functions \( f, g \), just as in the polynomial case. To show Lemma 3 in the case when \( p_i = a_i/b_i \) are rational functions, we note that \( \text{sign}(p_i(x)) = \text{sign}(a_i(x)b_i(x)) \), and furthermore \( \deg(a_i/b_i) \leq 2 \deg(p_i) \). Appealing to the polynomial version of Lemma 3 shows that it holds in the rational function setting with a slightly weaker upper bound \( K \leq 2(4emd/n)^2 \). This gets converted to a constant factor in the result of Theorem 3.

Next, we extend Lemma 1 by showing that generic LDR matrices have entries which are rational functions of their parameters. This immediately lets us conclude that neural networks built from any LDR matrices satisfy the VC dimension bounds of Theorem 3.

**Lemma 5.** If \( M \in \mathbb{R}^{m \times m} \) satisfies \( AM - MB = GH^T \), then the entries of \( M \) are polynomials of the entries of \( A, B, G, H \) with total degree at most \( c_1 m^{c_2} \) for some universal constants \( c_1, c_2 > 0 \).

**Proof.** The vectorization of the Sylvester equation \( AM - MB = GH^T \) is \( (I \otimes A - B^\top \otimes I) \text{vec}(M) = \text{vec}(R) \), where \( \text{vec} \) denotes the vectorization operation by stacking a matrix’s columns, and \( \otimes \) is the Kronecker product. Note that the entries of \( N^{-1} \) for an arbitrary matrix \( N \in \mathbb{R}^{n \times n} \) has degree \( n \) in the entries of \( N \), and \( R = GH^T \) has degree 2 in the entries of \( G, H \). Therefore the entries of

\[
\text{vec}(M) = (I \otimes A - B^\top \otimes I)^{-1} \text{vec}(R)
\]

have degree \( n^2 + 2 \) in the entries of \( A, B, G, H \). □

Note that many other classes of matrices satisfy this lemma. For example, a large class of matrices satisfying a property called low recurrence width was recently introduced as a way of generalizing many known structured matrices [14]. The low recurrence width matrices are explicitly defined through a polynomial recurrence and satisfy the bounded degree condition. Additionally, Lemma 5 holds when the parameters \( A, B \) themselves are structured matrices with entries having polynomial degree in terms of some parameters. This includes the case when they are quasiseparable matrices, the most general class of LDR previously analyzed [14].

### E Additional results

#### E.1 Additional baselines and comparisons at multiple parameter budgets

In Tables 6 and 7 we show the performance of our learned classes and baselines at parameter budgets corresponding to the LDR-TD and LDR-SD classes in the SHL and CNN models respectively.

On the MNIST-bg-rot dataset, we note that Chen et al. [7] also tested several methods on this dataset, including Random Edge Removal [11], Low Rank Decomposition [15], Dark Knowledge [22], HashedNets [7], and HashedNets with Dark Knowledge, and reported test errors of 73.17, 80.63, 79.03, 77.40, 59.20, and 58.25, where each method had 12406 parameters in the architecture. We found that our LDR-SD class, with 10986 parameters in the architecture, achieved a test error of 55.26.

Sindhwani et al. [40] later also tested on this dataset, and reported test errors of 68.4, 62.11, and 55.21 for Fastfood (10202 parameters), Circulant (8634 parameters), and Toeplitz-like, \( r = 2 \) (10986 parameters). Our method does exceed their reported results for Fastfood and Circulant [8], but not that of Toeplitz-like. We did find that we exceeded the performance of our own implementation of Toeplitz-like on this dataset (Table 1, Figure 4).

In Table 8 we also compare to network pruning [21], and show consistent improvements of LDR-SD over pruning at the same budget. We note that unlike the structured matrix methods, pruning requires first training the original model, followed by iterative retraining with a fixed sparsity pattern.

#### E.2 Sample complexity and generalization

As shown in Tables 10 and 11, we investigated how the performance of the structured and fully-connected layers varied with the amount of training data. On the MNIST-noise dataset, we trained both the single hidden layer and CNN models with random subsamples of 25%, 50%, and 75% of the training set, with 15% of the training set used for validation in all settings.
Table 6: Classification accuracy when replacing the hidden layer with structured classes in a single hidden layer architecture, at parameter budgets corresponding to LDR-TD and LDR-SD rank one. Rank is in parentheses. The first group of structured methods (in orange) all have compression factors (relative to fully-connected) of 98 on MNIST-bg-rot and MNIST-noise, and 128 on CIFAR-10 and NORB. The second group of structured methods (in blue) all have compression factors of 196 on MNIST-bg-rot and MNIST-noise, and 256 on CIFAR-10 and NORB.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST-bg-rot</th>
<th>MNIST-noise</th>
<th>CIFAR-10</th>
<th>NORB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully-connected</td>
<td>44.08</td>
<td>65.15</td>
<td>46.03</td>
<td>59.83</td>
</tr>
<tr>
<td>LDR-TD (r = 1)</td>
<td><strong>45.81</strong></td>
<td><strong>78.45</strong></td>
<td><strong>45.33</strong></td>
<td><strong>62.75</strong></td>
</tr>
<tr>
<td>Toeplitz-like [40] (r = 4)</td>
<td>42.67</td>
<td>75.75</td>
<td>41.78</td>
<td>59.38</td>
</tr>
<tr>
<td>Hankel-like (r = 4)</td>
<td>42.23</td>
<td>73.65</td>
<td>41.40</td>
<td>60.09</td>
</tr>
<tr>
<td>Vandermonde-like (r = 4)</td>
<td>37.14</td>
<td>59.80</td>
<td>33.93</td>
<td>48.98</td>
</tr>
<tr>
<td>Low-rank [15] (r = 4)</td>
<td>35.67</td>
<td>52.25</td>
<td>32.28</td>
<td>43.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST-bg-rot</th>
<th>MNIST-noise</th>
<th>CIFAR-10</th>
<th>NORB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR-SD (r = 1)</td>
<td><strong>44.74</strong></td>
<td><strong>78.80</strong></td>
<td><strong>43.29</strong></td>
<td><strong>63.78</strong></td>
</tr>
<tr>
<td>Toeplitz-like [40] (r = 2)</td>
<td>42.07</td>
<td>74.25</td>
<td>40.68</td>
<td>57.27</td>
</tr>
<tr>
<td>Hankel-like (r = 2)</td>
<td>41.01</td>
<td>71.20</td>
<td>40.46</td>
<td>57.95</td>
</tr>
<tr>
<td>Vandermonde-like (r = 2)</td>
<td>33.56</td>
<td>50.85</td>
<td>28.99</td>
<td>43.21</td>
</tr>
<tr>
<td>Low-rank [15] (r = 2)</td>
<td>32.64</td>
<td>38.85</td>
<td>24.93</td>
<td>37.03</td>
</tr>
</tbody>
</table>

Table 7: Classification accuracy when replacing the fully-connected layer with structured classes in a CNN architecture, at parameter budgets corresponding to LDR-TD and LDR-SD rank one. Rank is in parentheses. The first group of structured methods (in orange) all have compression factors (relative to fully-connected) of 98 on MNIST-bg-rot and MNIST-noise, and 128 on CIFAR-10 and NORB. The second group of structured methods (in blue) all have compression factors of 196 on MNIST-bg-rot and MNIST-noise, and 256 on CIFAR-10 and NORB.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST-bg-rot</th>
<th>MNIST-noise</th>
<th>CIFAR-10</th>
<th>NORB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully-connected</td>
<td>67.94</td>
<td>90.30</td>
<td>68.09</td>
<td>75.16</td>
</tr>
<tr>
<td>LDR-TD (r = 1)</td>
<td><strong>68.79</strong></td>
<td><strong>92.55</strong></td>
<td><strong>66.63</strong></td>
<td><strong>74.23</strong></td>
</tr>
<tr>
<td>Toeplitz-like [40] (r = 4)</td>
<td>63.23</td>
<td>91.60</td>
<td>67.10</td>
<td>72.25</td>
</tr>
<tr>
<td>Hankel-like (r = 4)</td>
<td>64.21</td>
<td>90.80</td>
<td>68.10</td>
<td>71.23</td>
</tr>
<tr>
<td>Vandermonde-like (r = 4)</td>
<td>61.76</td>
<td>90.40</td>
<td>63.63</td>
<td>72.11</td>
</tr>
<tr>
<td>Low-rank [15] (r = 4)</td>
<td>60.35</td>
<td>87.30</td>
<td>60.90</td>
<td>71.47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST-bg-rot</th>
<th>MNIST-noise</th>
<th>CIFAR-10</th>
<th>NORB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR-SD (r = 1)</td>
<td><strong>67.40</strong></td>
<td><strong>92.20</strong></td>
<td><strong>65.48</strong></td>
<td><strong>73.63</strong></td>
</tr>
<tr>
<td>Toeplitz-like [40] (r = 2)</td>
<td>63.63</td>
<td>91.45</td>
<td>67.15</td>
<td>71.64</td>
</tr>
<tr>
<td>Hankel-like (r = 2)</td>
<td>64.08</td>
<td>90.65</td>
<td><strong>67.49</strong></td>
<td>71.21</td>
</tr>
<tr>
<td>Vandermonde-like (r = 2)</td>
<td>51.38</td>
<td>86.50</td>
<td>58.00</td>
<td>68.08</td>
</tr>
<tr>
<td>Low-rank [15] (r = 2)</td>
<td>41.91</td>
<td>71.15</td>
<td>48.48</td>
<td>65.34</td>
</tr>
</tbody>
</table>

Table 8: On the MNIST-noise dataset, in the single hidden layer model, we compare to pruning \[21\] at multiple budgets. At each budget, we adjust the number of pruned weights to match the parameter budget of LDR-SD. As shown above, we find that the classification accuracy of LDR-SD consistently exceeds that of pruning.

<table>
<thead>
<tr>
<th>Rank of LDR-SD</th>
<th>LDR-SD</th>
<th>Pruning [21]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>77.91</strong></td>
<td>67.75</td>
</tr>
<tr>
<td>2</td>
<td><strong>77.95</strong></td>
<td>69.35</td>
</tr>
<tr>
<td>4</td>
<td><strong>78.32</strong></td>
<td>68.25</td>
</tr>
<tr>
<td>8</td>
<td><strong>78.63</strong></td>
<td>67.25</td>
</tr>
<tr>
<td>12</td>
<td><strong>78.33</strong></td>
<td>67.30</td>
</tr>
<tr>
<td>16</td>
<td><strong>78.08</strong></td>
<td>66.95</td>
</tr>
<tr>
<td>Method</td>
<td>MNIST-bg-rot</td>
<td>MNIST-noise</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>Fully-connected</td>
<td>55.78</td>
<td>21.63</td>
</tr>
<tr>
<td>LDR-TD</td>
<td>13.52</td>
<td>11.36</td>
</tr>
<tr>
<td>LDR-SD</td>
<td>12.87</td>
<td>12.65</td>
</tr>
<tr>
<td>Toeplitz-like</td>
<td>7.98</td>
<td>15.80</td>
</tr>
<tr>
<td>Low-rank</td>
<td>8.40</td>
<td>0.31</td>
</tr>
</tbody>
</table>

Table 9: Generalization error for fully-connected, LDR-TD, LDR-SD, Toeplitz-like, low-rank classes on the single hidden layer model. Consistent with Theorem 2, the structured classes have consistently lower generalization error than the fully-connected model. All are rank one.

<table>
<thead>
<tr>
<th>Method</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully-connected</td>
<td>59.30</td>
<td>61.85</td>
<td>65.35</td>
<td>65.15</td>
</tr>
<tr>
<td>LDR-TD</td>
<td>65.45</td>
<td>74.60</td>
<td>77.45</td>
<td>78.45</td>
</tr>
<tr>
<td>LDR-SD</td>
<td>67.90</td>
<td>71.15</td>
<td>76.95</td>
<td>78.80</td>
</tr>
<tr>
<td>Toeplitz-like</td>
<td>56.15</td>
<td>67.75</td>
<td>72.30</td>
<td>73.95</td>
</tr>
<tr>
<td>Low-rank</td>
<td>24.25</td>
<td>26.20</td>
<td>26.85</td>
<td>26.40</td>
</tr>
</tbody>
</table>

Table 10: On the MNIST-noise dataset, replacing the fully-connected layer with a structured layer reduces sample complexity in the single hidden layer model. Columns correspond to models trained on 25%, 50%, 75% and 100% of the training data (randomly subsampled). Both LDR-TD and LDR-SD only need 25% of the training data to exceed the performance of a fully-connected layer trained on 100% of the training data. All are rank one.

<table>
<thead>
<tr>
<th>Method</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully-connected</td>
<td>81.85</td>
<td>88.25</td>
<td>89.75</td>
<td>90.30</td>
</tr>
<tr>
<td>LDR-TD</td>
<td>86.45</td>
<td>91.35</td>
<td>93.00</td>
<td>92.55</td>
</tr>
<tr>
<td>LDR-SD</td>
<td>86.95</td>
<td>90.90</td>
<td>91.55</td>
<td>92.20</td>
</tr>
<tr>
<td>Toeplitz-like</td>
<td>81.65</td>
<td>88.15</td>
<td>90.90</td>
<td>90.95</td>
</tr>
<tr>
<td>Low-rank</td>
<td>33.15</td>
<td>38.40</td>
<td>42.55</td>
<td>44.55</td>
</tr>
</tbody>
</table>

Table 11: On the MNIST-noise dataset, replacing the fully-connected layer with a structured layer reduces sample complexity in the CNN model. Columns correspond to models trained on 25%, 50%, 75% and 100% of the training data (randomly subsampled). Both LDR-TD and LDR-SD only need 50% of the training data to exceed the performance of a fully-connected layer trained on 100% of the training data. All are rank one.

### E.3 Additional visualizations

In Figure 6 we visualize the learned subdiagonal on NORB along with images from the dataset.

### E.4 Rectangles dataset

Here we provide an example of a case where LDR-TD does not exceed the performance of the fixed operator classes in the single hidden layer model. In this dataset, created by Larochelle et al. [26], the task to classify, based on a binary image of a rectangle, whether the length is larger than the width. We show examples of the dataset in Figure 7. On this dataset, unlike the others, every structured class outperforms an unconstrained layer with 614656 parameters, including a circulant layer with only 784 parameters, and expanding the class beyond Toeplitz-like does not improve performance. We hypothesize this is because the approximate shift equivariance enforced by the Toeplitz-like class (discussed in Section 4 and Proposition 7 in Section C.3) is well suited for this dataset, and that further expansion of the class leads to overfitting.
Figure 6: As in Figure 5, we observed that the learned subdiagonal on the NORB dataset exhibits a circle-like pattern which roughly corresponds to the locations of objects of interest in the 2D inputs.

Figure 7: Examples of images from the rectangles dataset. [26]

<table>
<thead>
<tr>
<th>Method</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconstrained</td>
<td>91.94</td>
</tr>
<tr>
<td>LDR-TD (r = 1)</td>
<td>98.53</td>
</tr>
<tr>
<td>Toeplitz-like (r = 4)</td>
<td>99.29</td>
</tr>
<tr>
<td>Hankel-like (r = 4)</td>
<td>97.77</td>
</tr>
<tr>
<td>Vandermonde-like (r = 4)</td>
<td>94.11</td>
</tr>
<tr>
<td>Low-rank (r = 4)</td>
<td>92.80</td>
</tr>
<tr>
<td>NN [7]</td>
<td>91.91</td>
</tr>
<tr>
<td>Fastfood [44]</td>
<td>92.20</td>
</tr>
<tr>
<td>Circulant [8]</td>
<td>95.58</td>
</tr>
</tbody>
</table>

Table 12: Classification accuracy when replacing a single hidden layer with structured classes on the rectangles dataset [26]. Where applicable, rank (r) is in parentheses, and the number of parameters in the architecture is in italics below each method.
F Experimental details

F.1 Image classification

For all our experiments, batch sizes were chosen to be 50. NORB was downsampled to $32 \times 32$, and the left stereo image was used. Training was performed with stochastic gradient descent with momentum, with 50 epochs on all datasets. 15% of the training data was used for the validation set on all experiments. We fixed momentum at 0.9 for all methods for all experiments, and performed a grid search over learning rate. For each method, we tested the learning rates $5e^{-4}$, $2e^{-3}$, $1e^{-3}$, $2e^{-4}$. For each learning rate, we test on the validation set at each epoch, and report the test accuracy of the model with the highest validation accuracy.

In Figure 4, for each method and each of the four learning rates, we perform five trials with random initializations and report the average and standard deviation of the test accuracy of the learning rate with the highest average validation accuracy.

In these experiments, shown in Figure ?? in Appendix E, we tested on a LeNet-based architecture. The architecture has 2 convolution/pool layers with 6 and 16 channels respectively, followed by dense fully-connected layer, followed by fully connected logit/softmax layer. We replaced the second to last fully-connected layer, which was of dimensions $784 \times 784$ for the MNIST-bg-rot and MNIST-noise datasets, and $1024 \times 1024$ for the CIFAR-10 and NORB experiments.

Replacing convolutional layers In these experiments, shown in Figure ?? in Appendix E we tested on a LeNet-based architecture. The architecture has 2 convolution/pool layers with 6 and 16 channels respectively, followed by dense fully-connected layer, followed by fully connected logit/softmax layer. We replaced the second to last fully-connected layer, which was of dimensions $784 \times 784$ for the MNIST-bg-rot and MNIST-noise datasets, and $1024 \times 1024$ for the CIFAR-10 and NORB experiments.

Relevant neural architecture

In these experiments, shown in Figure ?? in Appendix E we tested on a LeNet-based architecture. The architecture has 2 convolution/pool layers with 6 and 16 channels respectively, followed by dense fully-connected layer, followed by fully connected logit/softmax layer. We replaced the second to last fully-connected layer, which was of dimensions $784 \times 784$ for the MNIST-bg-rot and MNIST-noise datasets, and $1024 \times 1024$ for the CIFAR-10 and NORB experiments.

We are interested in whether the successful and highly specific convolution layers of CNNs can be learned automatically. To investigate this, we test on the simplest possible multi-channel CNN model on the CIFAR-10 dataset. The model consists of one layer of convolution channels (3 RGB in channels, 3 out channels, stride 5), followed by a fully-connected layer and a final FC+softmax layer (total of 4 layers). We replace the convolutions with various structured matrices of the same dimensions, keeping the same $3 \times 3$ channel structure (e.g. it would consist of $3 \cdot 3 = 9$ square structured matrices) and number of hidden units.

The LDR classes benefit from being composed with LDR matrices of the same type (due to the composition property, Proposition 1(c)), so we additionally replace the later FC layer with the same structured matrix type.

By Proposition 1(d), channels of Toeplitz-like matrices form a larger Toeplitz-like matrix of the same size. Using this insight, we consider replacing the channel structure of the convolutional layer with either channels of structured matrices or a single wide structured matrix. (Also, note that this is able to leverage the asymptotic fast nature of our structured classes.)

Because it seems that convolution layers are strongly dependent on pooling – our structured matrices outperform them in isolation – we compare against a version of the CNN with an additional pooling layer after the convolutional channels. Note that this comparison is the same basic 4 layer model with a structured matrix vs. a 5 layer convolutional model with pooling. Since the architectures are quite different and hard to directly compare, we also experimented with adding more hidden units to the pooling model.

F.2 Language modeling

For a language modeling application, we explored replacing weight matrices in a recurrent neural network with structured matrices. We evaluate on a single layer LSTM architecture, defined by the update equations:

---

7 The convolutions have padding to ensure their input and output dimensions are equal.

8 Code available at https://github.com/pytorch/examples/tree/master/word_language_model
\[ i = \sigma(W_{ii}x + b_{ii} + W_{hi}h + b_{hi}) \\
\]

\[ f = \sigma(W_{if}x + b_{if} + W_{hf}h + b_{hf}) \\
\]

\[ g = \tanh(W_{ig}x + b_{ig} + W_{hg}h + b_{hg}) \\
\]

\[ o = \sigma(W_{io}x + b_{io} + W_{ho}h + b_{ho}) \\
\]

\[ c' = f \ast c + i \ast g \\
\]

\[ h' = o \tanh(c') \]

In our experiments we replace the matrices \( W_{ii}, W_{if}, W_{ig}, W_{io} \) with structured matrices. We use a hidden layer of size 128, and word embedding size of 128. We evaluate on the Wikitext-2 dataset, which consists of Wikipedia articles (2,088,628 training, 217,646 validation, and 245,569 test tokens). The total vocabulary is of size 33,278. We use the default hyperparameters and train using stochastic gradient descent with an initial learning rate of 20. The learning rate is annealed 4x after each epoch if performance does not improve on the validation set. Results are shown in Table 2.