2019 IEEE International Workshop on Signal Processing Systems

Structured Neural Network with Low Complexity for MIMO Detection

Siyu Liao*, Chunhua Deng*, Lingjia Liu† and Bo Yuan*
* Department of Electrical & Computer Engineering, Rutgers University
† Department of Electrical & Computer Engineering, Virginia Tech
Email: *siyu.liao@rutgers.edu, *chunhua.deng@rutgers.edu, †ljliu@vt.edu, *bo.yuan@soe.rutgers.edu

***Abstract*—Neural network has been applied into MIMO detection problem and has achieved the state-of-the-art performance. However, it is hard to deploy these large and deep neural network models to resource constrained platforms. In this paper, we impose the circulant structure inside neural network to generate a low complexity model for MIMO detection. This method can train the circulant structured network from scratch or convert from an existing dense neural network model. Experiments show that this algorithm can achieve half the model size with negligible performance drop.**

***Index Terms*—MIMO detection, neural network, circulant**

I. Introduction

With the prevalence of cellphones and tablets, mobile data transmission amount has been increasing in recent years [1]. However, due to lack of favorable radio spectrum frequency allocation, the wireless communication systems nowadays are constrained to support the huge data traffic [2]. In modern communication channels, it is now common to see the multiple input multiple output (MIMO) technology because of its high performance in terms of its spectral [3] and energy [4] efficiency.

This technology utilizes many antennas at transmitter and receiver side. As shown in Fig. 1, inputs with multiple symbols are transmitted simultaneously through the channel. Although there is usually some noise during the communication, receiver is required to detect or estimate the correct transmitted symbols. Each symbol can be detected individually or multiple symbols can be detected jointly. Joint detection is usually better in practice, but it involves higher computation complexity than individual symbol detection [2].

The optimal MIMO detection turns out to be NP-hard [5] which means the complexity of the problem can increase exponentially. For example, the maximum likelihood detector is optimal by minimizing the joint probability of detection errors. But its large computational complexity makes it hard to deploy in practice [6]. Therefore, there are many sub-optimal methods proposed for MIMO detection. One typical solution is using linear receivers, for example matched filters and zero forcing. Advanced methods include approximate message passing (AMP) [7], semi-definite relaxation [8] and so on. Particularly, AMP has been used in many scenarios, because it is simple and cheap to deploy while also providing a near optimal detection accuracy.

Machine learning has been found out to be effective in many applications such as computer vision [9] and traffic prediction [10]. Particularly, neural network is becoming an important approach to automatically learn from input by capturing hidden features of given data. This saves a lot of human efforts, especially the feature engineering work. The essential idea of neural network is to simulate neurons by activating nodes in the network and stacking multiple layers of such nodes. As activation goes through these layers, the input data is also transformed gradually into a high level representation. Eventually these representations or features can be used to solve some practical problems such as classification or regression.

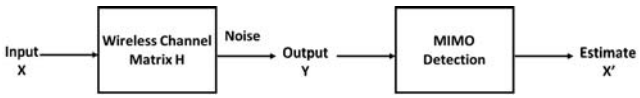

Fig. 1. Brief illustration of MIMO detection.

Neural network has also been proposed to address the MIMO detection problem and has achieved the state of the art performance [6]. Nowadays there is a trend towards making neural networks deeper with more and more layers, which usually achieves better performance in many applications such as ResNet [9]. It is found that deep neural networks are difficult to deploy in resource constrained platforms due to their large space and computation complexity [11]. After all, there can be millions of parameters within a neural network.

This paper aims at providing a low complexity neural network for MIMO detection problem. The essential idea is imposing the circulant structure for given neural network as in [12] to achieve a small and compact model. Deep neural network with circulant structure has been proven with the same theoretical guarantee as general neural network, i.e., the universal approximation property. More importantly, such structure is deployment friendly [13]. Because of its strong structure, the memory access has a strong pattern to follow. Moreover, note that matrix multiplication is one of the core computation inside deep neural network. The circulant structure can also speed up the general matrix multiplication with Fast Fourier Transform (FFT) when it comes to deployment in practice.

978-1-7281-1927-4/19/$31.00 ©2019 IEEE

II. Background

A. MIMO Detection

Generally speaking, the MIMO detection problem can be formulated as a linear system mapping n input to m output with some random noise. The input and output can take either real or complex values. Given that any linear mapping can be represented in a matrix form, the MIMO detection problem usually describes with channel matrix $\mathbf{H}$ as shown in Fig. 1.

A standard MIMO detection model can be described as following:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{w}, \tag{1}$$

where $\mathbf{y} \in \mathbb{C}^m$ is a vector of received signals, $\mathbf{H} \in \mathbb{C}^{m \times n}$ is channel matrix, $\mathbf{x} \in \mathbb{S}^n$ is the transmitted vector of identically independent distributed symbols from some constellation such as PSK. The $\mathbf{w} \in \mathbb{C}^m$ is a vector of noise following the standard Gaussian distribution. The goal of MIMO detection is to estimate input $\mathbf{x}$ given the received $\mathbf{y}$, which is described as $\mathbf{x}$' as in in Fig. 1.

It is assumed that we have the perfect channel state information (CSI) so the channel matrix $\mathbf{H}$ is already known. Although channel matrix is given, there is a difference between fixed channel and varying channel. Fixed channel means that $\mathbf{H}$ is always given as a constant matrix. Varying channel means that we only know the distribution of $\mathbf{H}$. A random channel matrix will be generated for transmitted symbols each time we detect. Thus, varying channel scenario is harder than fixed channel. In this paper, we mainly deal with varying channel.

B. Neural Network

Neural network contains multiple layers of neurons associated with activation functions. Typically nodes at each layer are only connected to nodes at the layer before and the layer after. Input is at the first layer, which will be passed to next layer after linear transform together with a nonlinear activation function. Following formulation presents a layer of neural network:

$$\mathbf{y} = f(\boldsymbol{\theta}\mathbf{x} + \mathbf{b}), \tag{2}$$

where $\mathbf{y} \in \mathbb{R}^m$ is the activation results of current layer, $f(\cdot)$ is the activation function, $\boldsymbol{\theta} \in \mathbb{R}^{m \times n}$ is a weight matrix representing the linear transform and $\mathbf{b} \in \mathbb{R}^n$ is a bias vector. Note that a neural network can stack multiple layers so as to become deeper. Last layer of the neural network presents the output.

Neural network training is often a supervised training process. All model parameters are learned from given data during training. Loss is computed between the network output and given ground truth results. The loss shall be minimized after the training phase, which is by using certain optimization method such as gradient descent. Therefore, usually the loss function is smooth and differentiable.

For example, the well-known back-propagation method [14] utilizes the first derivative of weight parameters and has been demonstrated effective in many applications. Its main idea is based on chain rule and gradient descent method:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \epsilon \frac{\partial L}{\partial \boldsymbol{\theta}_i}, \tag{3}$$

where ϵ is a scalar, $L(\cdot)$ is the loss function and i is the iteration number. To compute gradient of weight parameters between different layers, the chain rule will result in propagating gradients from a layer to its next layer. Since it is from the output layer to the input layer, it is then called back-propagation.

C. Neural Network for MIMO Detection

In this paper, we adopt the neural network structure as in [6] for MIMO detection problem, which is called DetNet. The network architecture is inspired by the projected gradient descent method and has achieved the state-of-the-art performance in MIMO detection. Projected gradient descent algorithm always chooses the closest projection after updating with gradient descent. The DetNet repeats this process with features at different layers. More specifically, the network is formulated as following:

$$\begin{aligned}
\mathbf{q}_i &= \mathbf{x}_{i-1} - \delta_{1,i}\mathbf{H}^T\mathbf{y} + \delta_{2,i}\mathbf{H}^T\mathbf{H}\mathbf{x}_{i-1} \\
\mathbf{z}_i &= f(\mathbf{W}_{1,i}\begin{bmatrix}\mathbf{q}_i \\ \mathbf{v}_{i-1}\end{bmatrix} + \mathbf{b}_{1,i}) \\
\mathbf{x}_i &= g(\mathbf{W}_{2,k}\mathbf{z}_i + \mathbf{b}_{2,i}) \\
\mathbf{v}_i &= \mathbf{W}_{3,i}\mathbf{z}_i + \mathbf{b}_{3,i} \\
\mathbf{x}_0 &= 0 \\
\mathbf{v}_0 &= 0
\end{aligned} \tag{4}$$

where i indicates the i-th layer, $f(\cdot)$ is an activation function, $g(\cdot)$ is an one-hot encoding function and model parameters include $\mathbf{W}_{1,i}$, $\mathbf{W}_{2,i}$, $\mathbf{W}_{3,i}$, $\mathbf{b}_{1,i}$, $\mathbf{b}_{2,i}$, $\mathbf{b}_{3,i}$. The $\delta_{1,i}$ and $\delta_{2,i}$ are pre-defined step size. The output of the network is $\mathbf{x}_L$ where L is the last layer number.

III. Related Work

Neural network has been found out with much redundancy in its weight parameters [15]. The difficulty of deploying large neural network intrigued many researchers. As a result, there are many methods for generating low complexity neural network, including pruning, low rank representation, regularization and structure transform based methods.

Pruning based methods essentially removes weight parameters from the network. For example, setting threshold [16] over the magnitude of parameters can effectively reduce the model size. Quantization is another method to compress neural networks, such as product quantization, residual quantization [17]. On the other hand, it is interesting to see that pruning weight parameters in frequency domain also works well in practice [18]. Although pruning based methods are simple and effective, the compressed neural network models are usually not deployment friendly due to their irregular weight distributions. More specifically, the memory access pattern is often random after using such methods.

Given that weight parameters of neural network are often found in the form of matrix or even tensor, a natural way to make a low complexity model is via low rank representation. Matrix decomposition methods like singular value decomposition (SVD) are useful in compressing neural networks [19]. Similarly, advanced tensor decomposition methods have also been applied to compress neural networks, such as tensor train [20] and Tucker decomposition [21]. These methods may reach high compression ratio but they require a long fine-tuning process to recover the model performance after compression.

Since training neural network is an optimization process, regularization is another straightforward approach to achieve small network models. Group lasso regularization [22] can make groups of weight parameters as zeros. This can avoid the disadvantage of random access as in pruning methods to some degree. Alternating direction method of multipliers (ADMM) is known as a powerful optimization algorithm. [23] proposes to replace L_0 norm with pruning and express in the ADMM form, which has achieved the state-of-the-art result. But this method takes a long training process since ADMM has multiple sub-problems to optimize.

Unlike methods aforementioned, it has been proven that enforcing structure to weight parameters can also approximate any continuous functions as general neural network [24]. This theoretical result gives a strong guarantee for structure matrix based methods. These methods turn out able to achieve high compression ratio in practice as well [25]. Imposing the structure matrix, it now has a regular pattern and is also with low space storage requirement. Moreover, structure matrix multiplication like circulant matrix can be accelerated via the Fast Fourier Transform.

IV. Methodology

In this section, we introduce the detailed circulant matrix based neural network as in [13]. This method is implemented and applied in our experiment, and turns out to be effective for MIMO detection problem.

A. Circulant Matrix Based Neural Network

A circulant matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$ is defined by a vector $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ as following:

$$\mathbf{W} = \begin{bmatrix} w_1 & w_n & \ldots & w_3 & w_2 \\ w_2 & w_1 & w_n & & w_3 \\ \vdots & w_2 & w_1 & \ddots & \vdots \\ w_{n-1} & & \ddots & \ddots & w_n \\ w_n & w_{n-1} & \ldots & w_2 & w_1 \end{bmatrix}. \tag{5}$$

It can be seen that values on diagonals are the same. Thus, the space complexity is linear $O(n)$ rather than $O(n^2)$ for general dense matrix. Moreover, the circulant matrix multiplication can be represented via FFT:

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \text{IFFT}(\text{FFT}(\mathbf{w}) \circ \text{FFT}(\mathbf{x})), \tag{6}$$

where $\mathbf{a} \in \mathbb{R}^n$ is the multiplication result, and $\mathbf{x} \in \mathbb{R}^n$ is the input vector. The $\circ$ stands for component-wise multiplication between two vectors. The computation complexity is now reduced to $O(n \log n)$. In summary, circulant matrix representation of weight matrix can lead to low complexity in terms of both space storage and computation time.

Given that most neural network training methods require first derivative of weight parameters, we present the calculation for the first derivative of circulant matrix:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{w}}, \tag{7}$$

where L is the loss function of the neural network and $\frac{\partial \mathbf{a}}{\partial \mathbf{w}} \in \mathbb{R}^{n\times n}$ is a Jacobian matrix. Fortunately, it is found that because of circulant structured weight matrix, this computation can also be accelerated via Fast Fourier Transform [13]:

$$\frac{\partial L}{\partial \mathbf{w}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}}) \circ \text{FFT}(\mathbf{x}')), \tag{8}$$

where $\mathbf{x}' = (x_1, x_n, x_{n-1}, \ldots, x_2)$.

According to the back-propagation algorithm [14], we also provide the first derivative of vector $\mathbf{a}$ so that it can be used for gradient computation of the other layer:

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{x}}, \tag{9}$$

where $\frac{\partial \mathbf{a}}{\partial \mathbf{x}} \in \mathbb{R}^{n\times n}$ is also an Jacobian matrix. Similarly, this can also be accelerated via Fast Fourier Transform [13]:

$$\frac{\partial L}{\partial \mathbf{x}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}}) \circ \text{FFT}(\mathbf{w}')), \tag{10}$$

where $\mathbf{w}' = (w_1, w_n, w_{n-1}, \ldots, w_2)$.

B. Block Circulant Based Neural Network

Note that circulant matrix is always a square matrix. In practice, neural network layer size can be arbitrarily determined according to different applications. Therefore, weight matrices are often not square as desired. To fit with a general setting of weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$, a straightforward design is to split weight matrix into square blocks. These blocks are set to be circulant matrix, and the entire matrix is then called block circulant matrix [13].

Let b be the block size and there are $\frac{m}{b} \times \frac{n}{b}$ blocks in total. Denote these block with $\mathbf{C}_{i,j} \in \mathbb{R}^{b\times b}$ for $i = 1, \ldots, \frac{m}{b}$ and $j = 1, \ldots, \frac{n}{b}$. Then weight matrix multiplication is reformulated as below:

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \begin{bmatrix} \mathbf{C}_{1,1} & \ldots & \mathbf{C}_{1,\frac{n}{b}} \\ \vdots & & \vdots \\ \mathbf{C}_{\frac{m}{b},1} & \ldots & \mathbf{C}_{\frac{m}{b},\frac{n}{b}} \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \ldots \\ \mathbf{a}_{\frac{m}{b}} \end{bmatrix}, \tag{11}$$

where $\mathbf{a}_i \in \mathbb{R}^b$ is a column vector. Since each $\mathbf{C}_{i,j}$ is a circulant matrix, we define $\mathbf{w}_{i,j}$ as the vector determining the matrix. Similarly, $\mathbf{a}_i$ can also be computed via FFT as aforementioned:

$$\mathbf{a}_i = \sum_{j=1}^{n/b} \mathbf{C}_{i,j}\mathbf{x}_j = \text{IFFT}(\sum_{j=1}^{n/b} \text{FFT}(\mathbf{w}_{i,j}) \circ \text{FFT}(\mathbf{x}_j)), \tag{12}$$

where $\mathbf{x}_j \in \mathbb{R}^b$ is a slice of vector $\mathbf{x}$ corresponding to the block $\mathbf{C}_{i,j}$.

Moreover, the first derivative of block circulant matrix is provided as follows:

$$\frac{\partial L}{\partial \mathbf{w}_{i,j}} = \frac{\partial L}{\partial \mathbf{a}_i}\frac{\partial \mathbf{a}_i}{\partial \mathbf{w}_{i,j}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{x}'_j)), \quad (13)$$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{x}_j} &= \sum_{i=1}^{m/b} \frac{\partial L}{\partial \mathbf{a}_i}\frac{\partial \mathbf{a}_i}{\partial \mathbf{x}_j} \\ &= \text{IFFT}(\sum_{i=1}^{m/b} \text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{w}'_{i,j}))\end{aligned}, \quad (14)$$

where $\mathbf{w}'_{i,j} \in \mathbb{R}^b$ and $\mathbf{x}'_j \in \mathbb{R}^b$ are defined similarly as in Eq. 8 and Eq. 10, respectively.

In summary, block circulant matrix takes $\frac{mn}{b}$ number of parameters and the multiplication complexity is $O(\frac{mn}{b}\log b)$. We also present the corresponding pseudocode for matrix multiplication and derivative computation as follows. The forward propagation algorithm is for block circulant matrix multiplication. The backward propagation algorithm is for computing the first derviative of block circulant matrix and the input vector.

Algorithm 1: Forward Propagation

Input: W, x
Output: a
partition $\mathbf{x}$ into n/b vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_{n/b}$;
partition $\mathbf{a}$ into m/b vectors, $\mathbf{a}_1, \ldots, \mathbf{a}_{m/b}$;
for $i \leftarrow 1$ *until* m/b **do**
  $\mathbf{a}_i \leftarrow 0$;
  for $j \leftarrow 1$ *until* n/b **do**
    $\mathbf{a}_i \leftarrow \mathbf{a}_i + \text{FFT}(\mathbf{w}_{i,j}) \circ \text{FFT}(\mathbf{x}_j))$;
  end
  $\mathbf{a}_i \leftarrow \text{IFFT}(\mathbf{a}_i)$;
end
return a;

Algorithm 2: Backward Propagation

Input: $\frac{\partial L}{\partial \mathbf{a}}, \mathbf{W}, \mathbf{x}$
Output: $\frac{\partial L}{\partial \mathbf{x}}, \frac{\partial L}{\partial \mathbf{W}}$
partition $\mathbf{x}$ into n/b vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_{n/b}$;
partition $\frac{\partial L}{\partial \mathbf{a}}$ into m/b vectors, $\frac{\partial L}{\partial \mathbf{a}_1}, \ldots, \frac{\partial L}{\partial \mathbf{a}_{m/b}}$;
for $j \leftarrow 1$ *until* n/b **do**
  $\frac{\partial L}{\partial \mathbf{x}_j} \leftarrow 0$;
  for $i \leftarrow 1$ *until* m/b **do**
    $\frac{\partial L}{\partial \mathbf{w}_{i,j}} \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{x}'_j))$;
    $\frac{\partial L}{\partial \mathbf{x}_j} \leftarrow \frac{\partial L}{\partial \mathbf{x}_j} + \text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{w}'_{i,j}))$;
  end
  $\frac{\partial L}{\partial \mathbf{x}_j} \leftarrow \text{IFFT}(\frac{\partial L}{\partial \mathbf{x}_j})$;
end
return $\frac{\partial L}{\partial \mathbf{x}}, \frac{\partial L}{\partial \mathbf{W}}$;

C. Circulant Approximation

Although we have described the training algorithm for circulant matrix based neural network, this requires training the entire network from scratch. In this section, we introduce another training option, i.e., converting from an existing dense neural network model to a circulant structured neural network model. Compared with training from scratch, this method can provide a warm up initialization which can converge faster than random initialization.

The essential idea is based on the circulant approximation algorithm as in [11]. For a dense matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$, let $\mathbf{w} \in \mathbb{R}^n$ be the vector for its closest circulant matrix. The optimal $\mathbf{w}$ can be found via following:

$$w_k = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{W}_{i,j} \times \mathbb{1}\left[\mathbf{W}_{i,j}\right]_{i-j \bmod n=k}, \quad (15)$$

where $w_k \in \mathbf{w}$ for $k = 1, \ldots, n$, and $\mathbb{1}\left[\cdot\right]$ is an indicator function. Here optimal is achieved under the Frobenius norm of the difference of given dense matrix and the approximated circulant matrix. Moreover, for the block circulant matrix, it is applicable to each block to achieve the approximation. The approximation algorithm is also summarized in pseudocode as below.

Algorithm 3: Circulant Approximation

Input: W
Output: w
initialize $\mathbf{w}$ as a zero vector;
for $i \leftarrow 1$ *until* n **do**
  for $j \leftarrow 1$ *until* n/b **do**
    $k \leftarrow (i-j) \bmod n$;
    $w_k \leftarrow w_k + \mathbf{W}_{i,j}$;
  end
end
for $i \leftarrow 1$ *until* n **do**
  $w_i \leftarrow w_i/n$;
end
return $\mathbf{w}$;

Instead of using an indicator function, it is easier to directly accumulate over the target weight parameter. Thus, using for loops to sweep over all possible indices can also give the circulant approximation. This algorithm can be applied to weight matrices inside a given pre-trained dense model and generate weight matrices for the target circulant structured model. As a result, this naive algorithm design has the $O(n^2)$ computation complexity.

D. Imposing Circulant Structure into DetNet

The DetNet architecture is mainly composed of fully connected layers, which is as described in Eq. 2. Thus, the circulant structure can be imposed into each weight matrix at each fully connected layer. More specifically, as shown in Eq. 4, all weight matrices including $\mathbf{W}_{1,i}$, $\mathbf{W}_{2,i}$, $\mathbf{W}_{3,i}$ can

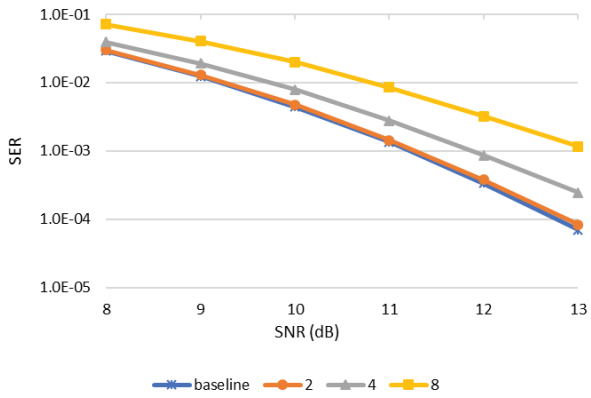

Fig. 2. QPSK model performance with different block size settings.

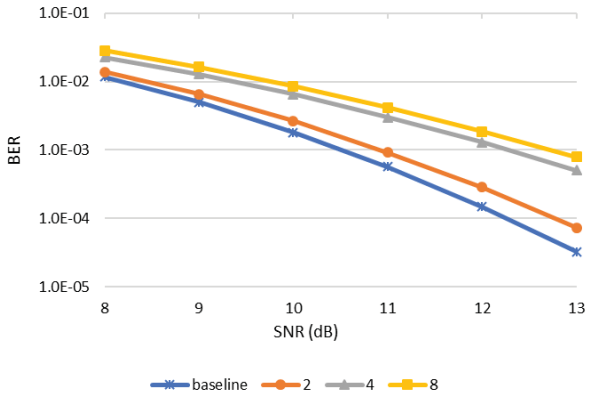

Fig. 3. BPSK model performance with different block size settings.

be with circulant structure. The compression ratio of a weight matrix can be adjusted simply by block size setting. However, note that there is a trade off between the compression ratio and the performance of the model with circulant structure.

V. Experiment

In this section, we provide experiment results with block circulant neural network over the MIMO detection model. We adopt the same model architecture as in [6] and apply the circulant structure to weight matrices. More specifically, all layers inside the neural network are set with the same block size in our experiment. We use the same re-parameterization method to handle complex variables with real values. The detection accuracy of hard decision output is evaluated under different signal-to-noise ratios (SNRs).

We adopt the varying channels settings as [6] in our experiment, including a 20×30 complex channel for QPSK and a 30×60 real channel for BPSK. The block size settings are 2, 4 and 8. The algorithm is implemented with Tensorflow [26] and runs in Ubuntu 16.04 with Intel(R) Xeon(R) CPU E5-2640 and Nvidia Tesla P100 GPU. We use the adam optimizer [27] in experiment with learning rate 0.0003, batch size 50, and 400K iterations.

Fig. 2 shows the result of QPSK performance for different settings. The baseline is the dense model performance. Block size settings with 2, 4 and 8 are found that with larger block size, the symbol error rate (SER) goes higher. As discussed in section IV, model size is linear to the block size. For example, model with block size 2 is half the size as baseline model. This indicates that with smaller model, the performance may get worse. It can also be seen that when SNR is small, performance of block size 2 and block size 4 are close but not block size 8. As SNR goes from small to large value, the performance discrepancy among different block size settings increase. The block size 2 is almost identical to the baseline under all SNRs.

Fig. 3 presents the result of BPSK performance under different block size settings. It is also found that larger compression ratio will result in worse model performance. When SNR is as small as 8, the model performance is always close to each other no matter the compression ratio. As SNR goes from small to large value, the performance discrepancy between baseline line and block size 2 slightly increase, so does the discrepancy between block size 4 and block size 8. However, the performance discrepancy between block size 2 and block size 4 will increase. As shown in the figure, when block size is 2, the model performance drop is negligible to the performance of baseline model.

VI. Conclusion

In this paper, we investigate the circulant structure network for MIMO detection problem. The algorithm is flexible to train from scratch or train from existing models. The block size setting can directly control the compressed model size. We explore different compression ratios to find the trade off between detection accuracy and model size. Experiment turns out that for QPSK we can achieve half the model size while maintaining almost identical performance. For BPSK, half the model size comes with a negligible accuracy drop.

Acknowledgment

This work is supported by National Science Foundation Awards CCF-1854742 and 1815699.

References

[1] J. Oueis and E. C. Strinati, "Uplink traffic in future mobile networks: Pulling the alarm," in *International Conference on Cognitive Radio Oriented Wireless Networks*. Springer, 2016, pp. 583–593.

[2] S. Yang and L. Hanzo, "Fifty years of mimo detection: The road to large-scale mimos," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 1941–1988, 2015.

[3] A. Paulraj, R. Nabar, and D. Gore, *Introduction to space-time wireless communications*. Cambridge university press, 2003.

[4] F. Rusek, D. Persson, B. K. Lau, E. G. Larsson, T. L. Marzetta, O. Edfors, and F. Tufvesson, "Scaling up mimo: Opportunities and challenges with very large arrays," *arXiv preprint arXiv:1201.3210*, 2012.

[5] S. Verdú, "Computational complexity of optimum multiuser detection," *Algorithmica*, vol. 4, no. 1-4, pp. 303–312, 1989.

[6] N. Samuel, T. Diskin, and A. Wiesel, "Learning to detect," *IEEE Transactions on Signal Processing*, vol. 67, no. 10, pp. 2554–2564, 2019.

[7] C. Jeon, R. Ghods, A. Maleki, and C. Studer, "Optimality of large mimo detection via approximate message passing," in *2015 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2015, pp. 1227–1231.
[8] W.-K. K. Ma, "Semidefinite relaxation of quadratic optimization problems and applications," *IEEE Signal Processing Magazine*, vol. 1053, no. 5888/10, 2010.
[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[10] S. Liao, L. Zhou, X. Di, B. Yuan, and J. Xiong, "Large-scale short-term urban taxi demand forecasting using deep learning," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 2018, pp. 428–433.
[11] S. Liao, A. Samiee, C. Deng, Y. Bai, and B. Yuan, "Compressing deep neural networks using toeplitz matrix: Algorithm design and fpga implementation," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 1443–1447.
[12] S. Liao, Z. Li, L. Zhao, Q. Qiu, Y. Wang, and B. Yuan, "Circconv: A structured convolution with low complexity," *arXiv preprint arXiv:1902.11268*, 2019.
[13] S. Liao, Z. Li, X. Lin, Q. Qiu, Y. Wang, and B. Yuan, "Energy-efficient, high-performance, highly-compressed deep neural network design using block-circulant matrices," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 458–465.
[14] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a backpropagation network," in *Advances in neural information processing systems*, 1990, pp. 396–404.
[15] Y. Izui and A. Pentland, "Analysis of neural networks with redundancy," *Neural Computation*, vol. 2, no. 2, pp. 226–238, 1990.
[16] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.
[17] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.
[18] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu, "Cnnpack: Packing convolutional neural networks in the frequency domain," in *Advances in neural information processing systems*, 2016, pp. 253–261.
[19] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition." in *Interspeech*, 2013, pp. 2365–2369.
[20] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in neural information processing systems*, 2015, pp. 442–450.
[21] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.
[22] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.
[23] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 184–199.
[24] L. Zhao, S. Liao, Y. Wang, Z. Li, J. Tang, and B. Yuan, "Theoretical properties for neural networks with weight matrices of low displacement rank," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 4082–4090.
[25] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan *et al.*, "C ir cnn: accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 395–408.
[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.
[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.