Learning Permutations in Monarch Factorization

Mimoun Mohamed
Aix Marseille Univ, CNRS, LIS, I2M
Marseille, France
mimoun.mohamed@lis-lab.fr

Valentin Emiya
Aix Marseille Univ, CNRS, LIS
Marseille, France
valentin.emiya@lis-lab.fr

Caroline Chaux
CNRS, IPAL
Singapore
caroline.chaux@cnrs.fr

***Abstract*—In order to reduce the quadratic cost of matrix-vector multiplications in dense and attention layers, Monarch matrices have been recently introduced, achieving a sub-quadratic complexity. It consists in factorizing a matrix using fixed permutations and learned block diagonal matrices, at the price of a small performance drop. We propose a more general model where some permutations are learned. The optimization algorithm explores the space of permutations using a Straight-Through Estimator (STE) inspired by the support exploration algorithm designed for sparse support recovery. Our experimental results demonstrate performance improvement in the context of sparse matrix factorization and of end-to-end sparse learning.**

***Index Terms*—monarch matrix, straight-through estimator, matrix factorization, sparse learning, permutation.**

I. Introduction

Matrices are ubiquitous in signal processing and machine learning, for data storage or linear mapping. Since they are responsible for computational and storage bottlenecks, extensive research has been dedicated to overcoming these limits.

Sparse Matrix Factorization. Sparse matrix factorization has been extensively studied for this purpose, with many models such as multilayer sparse matrix factorizations [1], [2] and Butterfly factorizations [3]–[5]. They are classes of structured matrices that can represent all relevant linear maps with low space and time complexity, along with efficient approximation and projection algorithms. More recently, the authors in [6] introduced the Monarch model, as a product of block-diagonal and permutation matrices. This model has been implemented in Monarch Mixer layers for neural networks. Both Butterfly and Monarch factorizations involve permutations, which are fixed [6] or chosen from a small candidate subset [4]. We propose to fully learn these permutations to further enhance the expressivity of sparse factorization models.

A Straight-Through Estimator to Learn Permutations. Learning permutations is a challenging task. General-purpose combinatorial algorithms, such as those involving the Sinkhorn operator [7], [8], require large datasets for training and are often task-specific. Relaxations, as they also need Sinkhorn operator [9], suffer from similar limitations. We propose to learn permutations using the Straight-Through Estimator (STE) principle [10], [11], a technique used to replace the gradient of a non-differentiable operator during backpropagation. STE is known for its empirical successes in binary and quantized neural networks [12], [13] and has been more recently applied to linear sparse problems [14]. When used to learn permutations for aligning multiple neural networks [15], it typically incurs a high computational overhead. In this paper, we propose an STE to learn permutations with low computational overhead.

Contributions and organization of the paper. After a brief technical introduction to Monarch matrices and STE in section II, section III presents the proposed generalized model and the corresponding iterative estimation procedure. To our knowledge, this is the first time an STE has been derived to learn permutations in a matrix factorization framework. The effectiveness of the proposed approach is demonstrated in section IV in two case studies: sparse matrix factorization, where the approach reduces the approximation error in dimensions up to 100; and end-to-end sparse learning, where our model is used as a layer in a neural network for classification, achieving a significant accuracy improvement over the existing Monarch factorization. We also provide details on the computational complexity and running times, showing that the performance gains are achieved with minimal computational overhead. Finally, conclusions are drawn in section V, including a discussion on the current work's limitations in learning permutations.

M. Mohamed was supported by a PhD grant from "Emploi Jeunes Doctorants (EJD)" plan which is funded by the French institution "Conseil régional Provence-Alpes-Côte d'Azur" and Euranova France. M. Mohamed gratefully acknowledges their financial support.

II. Preliminaries

A. Monarch factorization

Monarch matrices [6], [16] can replace dense matrices by a factorization that provides a lower time and space complexity when used for matrix multiplication, e.g., in dense and attention layers. In its most general form, an order-p Monarch matrix is structured as a product $M = \prod_{i=1}^{p} (P_i B_i) P_0$ that alternates permutations P_i and block-diagonal matrices B_i. In practice, existing works [6], [16] have focused on the model

$$M = P_2 L P_1 R P_0 \tag{1}$$

where $M \in \mathbb{R}^{N \times N}$, P_0, P_1, P_2 are fixed permutations, and $L, R \in \mathcal{BD}^{(N)}$ are block-diagonal matrices composed of n blocks of size $n \times n$, with $n^2 = N$. The model with $P_0 = I_N$ the identity matrix and $P_1 = P_2 = \bar{P}$ a 'base n' variant of the bit-reversal permutation has been studied in [6], while the model with $P_0 = P_1 = P_2 = \bar{P}$ has been used in [16]. In both cases, the resulting time and space complexity is in $\mathcal{O}\left(N^{\frac{3}{2}}\right)$.

For any matrix $A \in \mathbb{R}^{N\times N}$, [6] introduced an $\mathcal{O}(N^{\frac{5}{2}})$-time algorithm solving the projection problem on the set $\mathcal{M}^{(N)}$ of Monarch matrices with $P_0 = I_N$ and $P_1 = P_2 = \bar{P}$:

$$(\tilde{L}, \tilde{R}) = \pi_{\mathcal{M}}(A) \in \underset{L,R\,\in\,\mathcal{BD}^{(N)}}{\operatorname{argmin}} \frac{1}{2}\|\bar{P}L\bar{P}R - A\|_F^2 \quad (2)$$

We denote $\mathrm{p}_{\mathcal{M}}(A) = \bar{P}\tilde{L}\bar{P}\tilde{R}$ the resulting Monarch matrix. A similar projection addresses the case $P_0 = P_1 = P_2 = \bar{P}$ [16].

B. Straight-Through Estimator (STE)

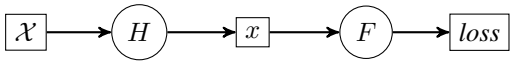
Let us consider an optimization problem of the form:

$$\underset{x\in\mathcal{S}}{\text{Minimize}}\, F(x) \quad (3)$$

where $F : \mathcal{D} \to \mathbb{R}$ is a differentiable function and $\mathcal{S}$ is a subset of the domain $\mathcal{D}$ of F, acting as a constraint on the solution, and making the optimization problem difficult. For instance, F may be a loss function, x the parameter vector to be learned, and $\mathcal{S}$ the set of vectors of $\mathcal{D}$ quantized on k-bit, or the set of k-sparse vectors in $\mathcal{D}$, for some integer k. Let us replace problem (3) by the unconstrained problem

$$\underset{\mathcal{X}\in\mathcal{D}}{\text{Minimize}}\, F(H(\mathcal{X})) \quad (4)$$

where $H : \mathcal{D} \to \mathcal{S}$ is a non-differentiable function. Assuming H is surjective, the minima of problems (3) and (4) coincide. One may address problem (4) using a descent strategy based on the computational graph



Since $\frac{\partial H}{\partial \mathcal{X}}$ is not defined, one may replace the gradient-descent update $\mathcal{X} \leftarrow \mathcal{X} - \eta\frac{\partial F\circ H}{\partial \mathcal{X}}|_{\mathcal{X}}$ by $\mathcal{X} \leftarrow \mathcal{X} - \eta\frac{\partial F}{\partial x}|_x$, for some learning rate $\eta > 0$. This comes from the approximation of the chain rule $\frac{\partial F\circ H}{\partial \mathcal{X}}|_{\mathcal{X}} = \frac{\partial F}{\partial x}|_x \frac{\partial H}{\partial \mathcal{X}}|_{\mathcal{X}} \approx \frac{\partial F}{\partial x}|_x$ known as the STE principle [10], [11]. It has been widely and successfully used for quantification in neural networks [12], [13] and more recently for sparsification in linear systems [14].

III. Proposed method

A. Generalizing Monarch Factorization

As mentioned in section II-A, the approximation of a matrix by a Monarch structure $P_2LP_1RP_0$ has only been performed with fixed permutations P_0, P_1, P_2 and by optimizing over block-diagonal matrices L and R. Our objective is to study the opportunity to optimize over some permutations too, to increase the expressivity of the model. Optimizing over a permutation is generally a difficult, combinatorial task. We propose to learn the permutations P_0 and P_2 while keeping P_1 fixed. By denoting by $\mathcal{P}_N$ the set of $N \times N$ permutation matrices, the optimization problem of interest writes:

$$\underset{\substack{L,R\,\in\,\mathcal{BD}^{(N)}\\ P_0,P_2\,\in\,\mathcal{P}_N}}{\text{Minimize}} \frac{1}{2}\|P_2L\bar{P}RP_0 - A\|_F^2. \quad (5)$$

We reformulate Problem (5) using Proposition 1, (keeping $P_1 = \bar{P}$ fixed to benefit from the projection $\mathrm{p}_{\mathcal{M}}$ onto the set of Monarch matrices):

Proposition 1. *Let $A \in \mathbb{R}^{N\times N}$ and define*

$$F : \begin{cases} \mathcal{P}_N \times \mathcal{P}_N & \to \mathbb{R} \\ (P_0, P_2) & \mapsto \frac{1}{2}\|\mathrm{p}_{\mathcal{M}}(\bar{P}P_2^TAP_0^T)\|_F^2 \end{cases}$$

Then, problem (5) *is equivalent to*

$$\underset{P_0,P_2\,\in\,\mathcal{P}_N}{\text{Minimize}}\, F(P_0, P_2). \quad (6)$$

Proof. Since $P_0, P_2, \bar{P}$ are orthogonal matrices, we have

$$\begin{aligned} &\min_{\substack{L,R\,\in\,\mathcal{BD}^{(N)}\\ P_0,P_2\,\in\,\mathcal{P}_N}} \frac{1}{2}\|P_2L\bar{P}RP_0 - A\|_F^2 \\ &\quad= \min_{\substack{L,R\,\in\,\mathcal{BD}^{(N)}\\ P_0,P_2\,\in\,\mathcal{P}_N}} \frac{1}{2}\|\bar{P}L\bar{P}R\bar{P} - \bar{P}P_2^TAP_0^T\|_F^2 \\ &\quad= \min_{P_0,P_2\,\in\,\mathcal{P}_N} F(P_0, P_2). \end{aligned}$$

□

B. Proposed algorithm

Our algorithm is an alternate scheme to minimize $F(P_0, P_2)$. To design an STE principle to optimize F over permutation matrices, Proposition 2 reformulates the problem as in (4), introducing dense variables $\mathcal{X}_0, \mathcal{X}_2$ and a mapping h.

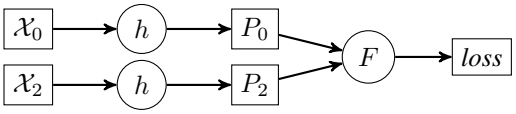
Proposition 2. *For $\mathcal{X}_0, \mathcal{X}_2 \in \mathbb{R}^{N\times N}$, let $H(\mathcal{X}_0, \mathcal{X}_2) = (h(\mathcal{X}_0), h(\mathcal{X}_2))$ with $h : \mathbb{R}^{N\times N} \to \mathcal{P}_N$ a surjective mapping. Then problem 6 is equivalent to*

$$\underset{\mathcal{X}_0,\mathcal{X}_2\,\in\,\mathbb{R}^{N\times N}}{\text{Minimize}}\, F(H(\mathcal{X}_0, \mathcal{X}_2)) \quad (7)$$

Proof. Since h is surjective, the image of H is $\mathcal{P}_N \times \mathcal{P}_N$ and the minima of F on $\mathcal{P}_N \times \mathcal{P}_N$ and of $F \circ H$ coincide. □

By considering $\mathcal{X}_0$ and $\mathcal{X}_2$ as cost matrices, h can be chosen as any algorithm solving the well-known linear assignment problem. We use the aggressive auction algorithm [17] for its recognized performance. It is a surjective mapping since $h(P) = P$ for any $P \in \mathcal{P}_N$.

We propose algorithm 1 called STE Alternate Minimization for Monarch (STEAM4Monarch) to compute an approximate solution of problem (7). It implements an STE-based back-propagation procedure on the computational graph



STEAM4Monarch alternates between STE-based updates of $\mathcal{X}_i$ for $i \in \{0, 2\}$ (line 9) with a step size controlled by the Lipschitz modulus of the gradient (line 8). This update is similar to a gradient step where the gradient $\frac{\partial F\circ H}{\partial \mathcal{X}_i}|_{(\mathcal{X}_0,\mathcal{X}_2)} = \frac{\partial F}{\partial P_i}|_{(P_0,P_2)}\frac{\partial h}{\partial \mathcal{X}_i}|_{\mathcal{X}_i}$ is approximated by:

$$\begin{aligned} \frac{\partial F}{\partial P_i}|_{(P_0,P_2)} &= \frac{\partial}{\partial P_i}\frac{1}{2}\|P_2L\bar{P}RP_0 - A\|_F^2 \\ &= \begin{cases} (P_2L\bar{P}R)^T(P_2L\bar{P}RP_0 - A) & \text{if } i = 0, \\ (P_2L\bar{P}RP_0 - A)(L\bar{P}RP_0)^T & \text{if } i = 2. \end{cases} \end{aligned}$$

Permutation P_i is then computed from $\mathcal{X}_i$ (line 10) and block-diagonal factors L and R are obtained from the Monarch projection (line 11) based on proposition 2 and problem (6).

A natural initialization consists in setting $P_0, P_2, \mathcal{X}_0, \mathcal{X}_2$ to $\bar{P}$ and in using the Monarch projections for L and R (lines 4-5). The best variables encountered through the procedure are returned (line 16) since the objective function may not decrease at each iteration. Indeed, as detailed in [14] for sparse recovery, we have exploratory variables $\mathcal{X}_i$ that accumulate gradients along iterations, which is useful to explore the set of permutations and to escape from local minima. A typical behavior is represented in fig. 1.

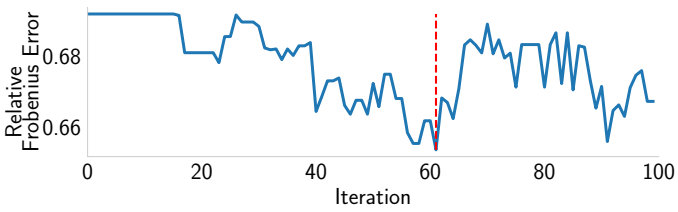


Fig. 1. Relative Frobenius error as a function of the iteration, in the setting of section IV-A, with $N = 49$. The dashed red line indicates t_{BEST}.

Algorithm 1 STEAM4Monarch

1: **Input:** Matrix A
2: **Output:** Factors $\tilde{P}_2, \tilde{L}, \tilde{R}, \tilde{P}_0$ of $\hat{A} = \tilde{P}_2\tilde{L}\bar{P}\tilde{R}\tilde{P}_0$
3: $t \leftarrow 1$
4: $P_0, P_2, \mathcal{X}_0, \mathcal{X}_2 \leftarrow \bar{P}$
5: $(L, R) \leftarrow \pi_{\mathcal{M}}(AP_0^T)$
6: **loop** T times
7: **for** $i \in \{0, 2\}$ **do**
8: $\eta \leftarrow \frac{1}{\alpha\|L\bar{P}R\|_2^2}$
9: $\mathcal{X}_i \leftarrow \mathcal{X}_i - \eta \frac{\partial}{\partial P_i} \frac{1}{2}\|P_2 L\bar{P}RP_0 - A\|_F^2$
10: $P_i \leftarrow h(\mathcal{X}_i)$
11: $(L, R) \leftarrow \pi_{\mathcal{M}}(\bar{P}P_2^T AP_0^T)$
12: **end for**
13: $L^t, R^t, P_0^t, P_2^t \leftarrow L, R, P_0, P_2$
14: $t \leftarrow t + 1$
15: **end loop**
16: $t_{BEST} = \underset{t' \in [\![1,t]\!]}{\operatorname{argmin}} \|P_2^{t'} L^{t'} \bar{P} R^{t'} P_0^{t'} - A\|_F$
17: **Return:** $P_2^{t_{BEST}}, L^{t_{BEST}}, R^{t_{BEST}}, P_0^{t_{BEST}}$

Each iteration of our algorithm involves a few matrix multiplications, with more than half involving permutations, with a matrix multiplication time complexity of $\mathcal{O}(N^2)$. The remaining products can be rearranged to include a block-diagonal factor, leading to a time complexity of $\mathcal{O}(N^{5/2})$. The time complexity of the projection $\pi_{\mathcal{M}}$ is $\mathcal{O}(N^{5/2})$. The best theoretical worst-case complexity of an assignment algorithm [18] is $\mathcal{O}(N^{5/2} \log NC)$, $C = \|\mathcal{X}\|_\infty$. However [19] shows that the experimental complexity of the auction algorithm is typically lower, with its best-case complexity being $\mathcal{O}(N^2)$. Furthermore, after the first iteration, we initialize the auction algorithm with elements from its previous instance. It is known [20] that such a strategy significantly reduces computation time. Thus, the overall running time of our algorithm largely depends on the empirical performance of the auction algorithm. In the worst case it stays sub-cubic, like Monarch projection.

IV. Numerical experiment

We study the performance of our approach against the original Monarch factorization with fixed permutations. First, we examine the ability of models and algorithms to fit a matrix generated from the true model (1). Then, we illustrate their learning capacity in a supervised learning task. Experiments were conducted on a single CPU, with Python code available at https://gitlab.lis-lab.fr/valentin.emiya/steam4monarch.

A. Sparse matrix factorization

In this experiment, we randomly generate matrices M from the true model eq. (1). The block-diagonal entries of L and R are drawn independently from the standard normal distribution; P_2 is uniformly drawn from $\mathcal{P}_N$; and P_0 is either set to I_N, or uniformly drawn from $\mathcal{P}_N$. We vary the matrix size $N \in \{n^2 \,|\, n \in [\![2, 10]\!]\}$, with $r = 1000$ runs for each value of N. Solutions are estimated from the original Monarch projection; from the proposed Algorithm 1 (skipping the inner loop for $i = 0$ in the case $P_0 = I_N$), with $\alpha = 1.001$ and $T \in \{100, 1000\}$ iterations; and from a pure random exploration strategy. In the latter, random permutations are uniformly drawn T times from $\mathcal{P}_N$, followed by a Monarch projection to estimated L and R, and the best encountered factorization is returned.

The performance is reported in fig. 2 using the Relative Frobenius Error $\frac{\|\hat{M}-M\|_F}{\|M\|_F}$ between M and the approximation $\hat{M}$ returned by each algorithm, as a function of N. Our algorithm significantly outperforms the Monarch projection for all considered matrix sizes N. The largest difference is observed in the lower dimensions while increasing the number of iterations in our algorithm improves its performance. The pure random strategy is not represented since it achieves very poor performance, above 1 almost everywhere. We have also conducted a similar factorization experiment where the observed matrix has i.i.d. gaussian entries: results are similar as in fig. 2 with better approximation performance for the proposed approach. This experiment suggests that our approach is able to provide an better factorization than the Monarch projection but that exploring the permutation space is a difficult task. Indeed, our algorithm does not generally retrieve the best permutation, it performs better when one permutation is searched (case $P_0 = I_N$) than in the more difficult case with two unknown permutations. Yet it is able to find good permutations compared to a pure exploration strategy that fails to return interesting factorizations.

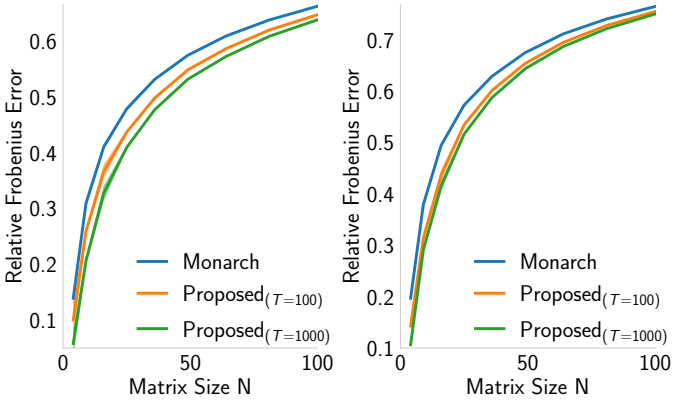


Fig. 2. Relative Frobenius Error for various factorizations as a function of the matrix size N. M is randomly generated from eq. (1) with $P_0 = I_N$ (left) or $P_0 \in \mathcal{P}_N$ (right). 99% confidence interval are very narrow (shaded areas, hardly visible when zooming).

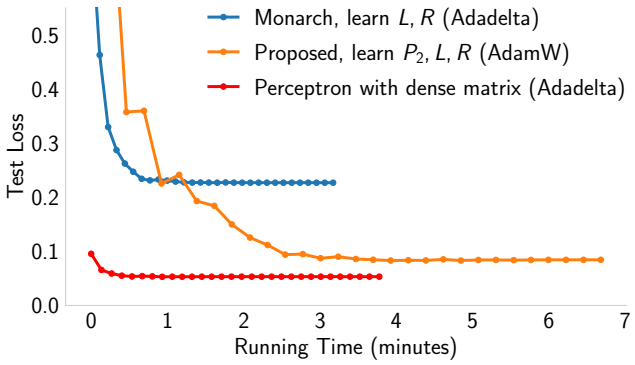
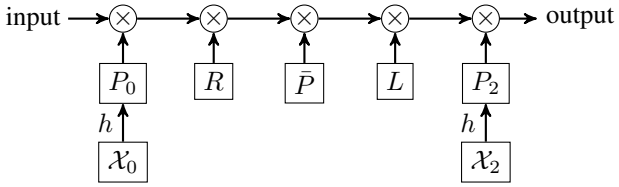


Fig. 3. Test loss on MNIST dataset over running time of the training stage for a Perceptron with a hidden layer made of a dense matrix (615k parameters), a Monarch factorization (45k parameters), or our factorization (training: 659k parameters, inference: 45k parameters). Each point represents one epoch.

B. End-to-end sparse learning

To compare the learning capability of our model with the Monarch model, we propose an end-to-end learning setting inspired from [6]. We perform the classification task on the MNIST dataset [21] with the original train/test split, using a multilayer perceptron (MLP) with a hidden layer of size 784, a ReLU activation, and a LogSoftmax decision layer before computing a negative log-likelihood loss. We train this MLP using first-order optimizers Adadelta [22] and AdamW [23] for 30 epochs. Then, we repeat the training process from scratch by replacing the 784×784 dense matrix of the hidden layer either by a Monarch factorization or by the proposed factorization, represented by the computational graph



Since there is no observed matrix to be factorized, the Monarch projection and Algorithm 1 are not applicable. However, the STE can be used to backpropagate gradients through h and our factorization – and the Monarch factorization – can be trained in the same manner as the MLP. We focus on cases where we learn, along with L and R, either P_0 and/or P_2. When learned, P_0 and P_2 are initialized to I_N. Otherwise, they are set to $\bar{P}$.

TABLE I
TEST ACCURACY ON MNIST WITH ADAMW AND ADADELTA

	Adadelta	**AdamW**
Dense matrix	98.41%	96.35%
Monarch (learn L, R)	96.26%	92.92%
Proposed (learn P_2, L, R)	96.12%	**97.49%**
Proposed (learn L, R, P_0)	**96.39%**	97.00%
Proposed (learn P_2, L, R, P_0)	96.38%	96.08%

Table I shows the test accuracy of each model after training. While the accuracy of the Monarch model with fixed permutation (96.26%) drops by more than two points compared to the MLP (98.41%), learning one permutation achieves a better performance (97.49%), only one point below the MLP. Also, our model is less sensitive to the choice of the optimizer. As in the factorization experiment, our model performs better when only one permutation is learned. Figure 3 presents the test loss over running time. While our code is not fully optimized, our model has similar performance as the original Monarch one at time $t = 1$ when the loss of the latter plateaus, even if each epoch takes more time for our model. A limited computational overhead can further be used by our approach to improve significantly the loss close to the MLP performance. This is of particular interest at inference time where space and time complexities are proportional to the number of learned parameters in this layer, which decreases from $N^2 = 614,656$ for the MLP to $2N^{\frac{3}{2}} + N = 44,688$ for our approach and $2N^{\frac{3}{2}} = 43,904$ for the original Monarch model.

V. CONCLUSIONS AND PERSPECTIVES

We have extended the Monarch model to improve its expressivity, by learning some permutations via an original STE principle. We have designed an algorithm to estimate such a factorization and have also shown how to learn it as a layer in a neural network in an end-to-end procedure. The proposed experiments have demonstrated the efficiency of the approach while keeping a limited computational overhead.

Our study also illustrates the challenges of learning permutations and highlights the limits of the approach. The estimated permutations are not necessarily optimal, and higher performance could be achieved by improving the optimization procedure. Learning multiple permutations simultaneously proves to be even more difficult. Also, due to the use of matrices $\mathcal{X}_i$, the training procedure has quadratic complexity compared to the sub-quadratic one when permutations are fixed. Eventually, monarch models with orders higher than 2 require further investigation.

REFERENCES

[1] L. Le Magoarou and R. Gribonval, "Flexible multilayer sparse approximations of matrices and applications," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 4, pp. 688–700, 2016.

[2] Q.-T. Le and R. Gribonval, "Structured Support Exploration For Multilayer Sparse Matrix Factorization," in *Proc. Int. Conf. Acoust. Speech Signal Process.*, Jun. 2021.

[3] Y. Li, H. Yang, E. Martin, K. L. Ho, and L. Ying, "Butterfly factorization," *Multiscale Model. Simul.*, vol. 13, pp. 714–732, 2015.

[4] T. Dao, A. Gu, M. Eichhorn, A. Rudra, and C. Re, "Learning fast algorithms for linear transforms using butterfly factorizations," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2019.

[5] T. Dao, N. Sohoni, A. Gu, M. Eichhorn, A. Blonder, M. Leszczynski, A. Rudra, and C. Ré, "Kaleidoscope: An efficient, learnable representation for all structured linear maps," in *Eighth International Conference on Learning Representations*, 2020.

[6] T. Dao, B. Chen, N. S. Sohoni, A. Desai, M. Poli, J. Grogan, A. Liu, A. Rao, A. Rudra, and C. Ré, "Monarch: Expressive structured matrices for efficient and accurate training," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2022, pp. 4690–4721.

[7] G. Mena, D. Belanger, S. Linderman, and J. Snoek, "Learning latent permutations with gumbel-sinkhorn networks," *arXiv preprint arXiv:1802.08665*, 2018.

[8] P. Emami and S. Ranka, "Learning permutations with sinkhorn policy gradient," *arXiv preprint arXiv:1805.07010*, 2018.

[9] J. Lyu, S. Zhang, Y. Qi, and J. Xin, "Autoshufflenet: Learning permutation matrices via an exact lipschitz continuous penalty in deep convolutional neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 608–616.

[10] G. E. Hinton, "Neural networks for machine learning," Coursera, video lectures, 2012, lecture 15b.

[11] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, vol. arXiv:1308.3432, 2013.

[12] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Adv. Neural Inf. Process. Syst.*, vol. 28, Montreal, Quebec, Canada, Dec. 7–12 2015, pp. 3123–3131.

[13] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[14] M. Mohamed, F. Malgouyres, V. Emiya, and C. Chaux, "Straight-through meets sparse recovery: the support exploration algorithm," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2024.

[15] S. K. Ainsworth, J. Hayase, and S. Srinivasa, "Git re-basin: Merging models modulo permutation symmetries," *arXiv preprint arXiv:2209.04836*, 2022.

[16] D. Fu, S. Arora, J. Grogan, I. Johnson, E. S. Eyuboglu, A. Thomas, B. Spector, M. Poli, A. Rudra, and C. Ré, "Monarch mixer: A simple sub-quadratic gemm-based architecture," *Adv. Neural Inf. Process. Syst.*, vol. 36, 2024.

[17] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Ann. Oper. Res.*, vol. 14, no. 1, pp. 105–123, 1988.

[18] J. B. Orlin and R. K. Ahuja, "New scaling algorithms for the assignment and minimum mean cycle problems," *Math. Programm.*, vol. 54, no. 1, pp. 41–56, 1992.

[19] C. A. Alfaro, S. L. Perez, C. E. Valencia, and M. C. Vargas, "The assignment problem revisited," *Optim. Lett.*, vol. 16, no. 5, pp. 1531–1548, 2022.

[20] D. Bertsekas, "New auction algorithms for the assignment problem and extensions," *Results in control and optimization*, vol. 14, p. 100383, 2024.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] M. Zeiler, "Adadelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[23] I. Loshchilov, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.