24-29 July, 2023,
Honolulu, Hawaii

PMaF: Deep Declarative Layers for Principal Matrix Features

Zhiwei Xu, Hao Wang, Yanbin Liu, Stephen Gould
zhiwei.xu@anu.edu.au

Workshop on Differentiable Almost Everything: Differentiable Relaxations, Algorithms, Operators, and Simulators

Motivation

Principal matrix feature (PMaF): a single vector summarising a data matrix.

- Useful for feature representation with a lower dimension (a vector)
- Better and possibly faster solutions that are constrained on a sphere
- Differentiable for end-to-end learning
- Implicit differentiation [1] with higher running speed and lower hardware memory requirements than unrolling the forward optimization iteration [2] or without exploited structures [3]

Two Deep Declarative Layers

1. Least Squares on Sphere (LESS)

$$\begin{aligned}&\text{Given } \mathbf{A}\in\mathbb{R}^{m\times n} \text{ and } \mathbf{b}\in\mathbb{R}^{m},\\ &\text{minimize}_{\mathbf{u}\in\mathbb{R}^n}\ \mathtt{f}(\mathbf{A},\mathbf{b},\mathbf{u}) \triangleq \frac{1}{2}\|\mathbf{A}\mathbf{u}-\mathbf{b}\|^2\\ &\text{subject to } \|\mathbf{u}\|^2=1\end{aligned}$$

Iterative optimization:

1. Projected gradient descent (PGD)
2. + Direction weight (DW)

$$w_t = 1-\mathtt{S_c}(\mathbf{d}_t,\mathbf{u}_0),\quad \mathtt{S_c}(\mathbf{a},\mathbf{b}) = \mathbf{a}^T\mathbf{b}/\left(\|\mathbf{a}\|\,\|\mathbf{b}\|\right),\quad \mathbf{d}_t=\begin{cases}-\nabla\mathtt{f}(\mathbf{u}_t) & \text{if } \|\mathbf{u}_0\|\geq 1\\ \nabla\mathtt{f}(\mathbf{u}_t) & \text{otherwise}\end{cases}$$

3. + Riemannian manifold (RM)

$$\mathtt{Proj_{RM}}\left(-\nabla\mathtt{f}(\mathbf{u}_t)\right) = \left(\mathbf{I}_n-\mathbf{u}_t\mathbf{u}_t^T\right)\left(-\nabla\mathtt{f}(\mathbf{u}_t)\right)$$

$$\mathbf{u}_{t+1} = \mathtt{Proj_{Sph}}\left(\mathbf{u}_t+\eta\mathtt{Proj_{RM}}\left(-\nabla\mathtt{f}(\mathbf{u}_t)\right)\right)$$
$$\eta\leftarrow\beta\eta$$

4. + Backtracking line search (BLS)

$$\mathtt{f}(\mathbf{u}_t+\eta\Delta\mathbf{u}_t)\leq\mathtt{f}(\mathbf{u}_t)+\alpha\eta\nabla^T\mathtt{f}(\mathbf{u}_t)\Delta\mathbf{u}_t$$

5. + Tangent weight decay (TWD)

$$\mathtt{S_c}\left(\mathtt{Proj_{RM}}\left(-\nabla\mathtt{f}(\mathbf{u}_t)\right),\mathtt{Proj_{RM}}\left(-\nabla\mathtt{f}(\mathbf{u}_{t+1})\right)\right)<0$$

2. Implicit Eigen Decomposition (IED)

$$\begin{aligned}&\text{Given } \mathbf{A}\in\mathbb{R}^{m\times m},\\ &\text{minimize}_{\mathbf{u}\in\mathbb{R}^{m\times n}}\ \mathtt{f}(\mathbf{A},\mathbf{u})\triangleq-\mathtt{tr}\left(\mathbf{u}^T\mathbf{A}\mathbf{u}\right)\\ &\text{subject to }\ \mathtt{h}(\mathbf{u})\triangleq\mathbf{u}^T\mathbf{u}=\mathbf{I}_n\end{aligned}$$

Iterative optimization:

1. Power iteration (PI)

$$\mathbf{u}_{t+1}=\mathbf{A}\mathbf{u}_t/\|\mathbf{A}\mathbf{u}_t\|$$
$$\mathbf{y}=\mathbf{u}_K \quad\text{and}\quad \lambda=\mathbf{y}^T\mathbf{A}\mathbf{y}$$

2. Simultaneous iteration (SI)

$$\{\mathbf{Q}_t,\mathbf{R}_t\}=\mathtt{QR}(\mathbf{x}_t)\quad\text{and}\quad\mathbf{x}_{t+1}=\mathbf{x}_t\mathbf{Q}_t$$
$$\mathbf{y}=\text{the component of }\mathbf{Q}_K\text{ to }\lambda=\mathtt{max}(\mathbf{R}_K)$$

➤ *Solution consistency in iterations*

$$\textit{Historical}:\quad \mathbf{u}_t\leftarrow\mathtt{V}(\mathbf{u}_t,\mathbf{u}_{t-1})\,\mathbf{u}_t$$
$$\textit{Hard-coded}:\quad \mathbf{y}\leftarrow\mathtt{V}(\mathbf{y},\mathbf{r})\,\mathbf{y}$$
$$\mathtt{V}(\mathbf{a},\mathbf{b})=\mathtt{Sign}\left(\mathbf{a}^T\mathbf{b}\right)\text{ if }(\mathbf{a}\not\perp\mathbf{b})\text{ and otherwise }1$$

Deep Declarative Networks as Backward

Implicit differentiation of the learning loss over inputs using DDN [1] and exploited structures [3]:

$$\nabla_X L=\nabla_Y L\left(\mathcal{H}^{-1}\mathcal{A}^T\left(\mathcal{A}\mathcal{H}^{-1}\mathcal{A}^T\right)^{-1}\mathcal{A}-\mathbf{I}_n\right)\mathcal{H}^{-1}\mathcal{B}$$

LESS

Vanilla

$$\mathcal{A}=2\mathbf{y}^T\in\mathbb{R}^{1\times n},$$
$$\mathcal{B}=\nabla^2_{XY}\mathtt{f}(\mathbf{A},\mathbf{b},\mathbf{y})\in\mathbb{R}^{n\times(m\times n)}$$
$$\mathcal{H}=\mathbf{A}^T\mathbf{A}-2\beta\mathbf{I}_n\in\mathbb{R}^{n\times n}$$
$$\beta=\frac{1}{2}\mathbf{y}^T\mathbf{A}^T\left(\mathbf{A}\mathbf{y}-\mathbf{b}\right)\in\mathbb{R}$$

With exploited structures

$$\mathcal{B}_{ij}=\mathbf{A}_i\mathbf{y}_j^T,\quad\forall i,j\in\mathcal{N},$$
$$\mathcal{B}_{ii}\leftarrow\mathcal{B}_{ii}+(\mathbf{A}\mathbf{y}-\mathbf{b}),\quad\forall i\in\mathcal{N}.$$

IED

Vanilla

$$\mathcal{A}=2\mathbf{y}^T\in\mathbb{R}^{1\times m}$$
$$\mathcal{B}=\nabla^2_{XY}\mathtt{f}(\mathbf{A},\mathbf{y})\in\mathbb{R}^{m\times(m\times m)}$$
$$\mathcal{H}=-\left(\mathbf{A}+\mathbf{A}^T\right)-2\beta\mathbf{I}_m\in\mathbb{R}^{m\times m}$$
$$\beta=-\frac{1}{2}\mathbf{y}^T\left(\mathbf{A}\mathbf{y}+\mathbf{A}^T\mathbf{y}\right)\in\mathbb{R}$$

With exploited structures

$$\mathcal{B}_{ijk}=0,\quad\forall i,j,k\in\mathcal{M},$$
$$\mathcal{B}_{iji}\leftarrow\mathcal{B}_{iji}-\mathbf{y}_j,\quad\forall i,j\in\mathcal{M},$$
$$\mathcal{B}_{iij}\leftarrow\mathcal{B}_{iij}-\mathbf{y}_j,\quad\forall i,j\in\mathcal{M}.$$

Further $\nabla_X L=-\mathcal{K}\mathbf{y}^T-\mathbf{y}\mathcal{K}^T.$

IED-implicit function theorem (IFT)

Objective function

$$\mathtt{f}(\mathbf{A},\mathbf{y})=\mathbf{y}-\mathbf{A}\mathbf{y}/\|\mathbf{A}\mathbf{y}\|$$

Implicit gradients

$$\nabla_X\mathbf{y}=-\left(\nabla_Y\mathtt{f}(\mathbf{A},\mathbf{y})\right)^{-1}\nabla_X\mathtt{f}(\mathbf{A},\mathbf{y})$$
$$\nabla_X L=\nabla_Y L\nabla_X\mathbf{y}=-\nabla_Y L\mathcal{H}^{-1}\mathcal{B}$$
$$\mathcal{H}=\mathbf{I}_m-\left(\mathbf{I}_m-\mathbf{y}\mathbf{y}^T\right)\mathbf{A}/\lambda,$$
$$\mathcal{B}=-\left(\mathbf{I}_m-\mathbf{y}\mathbf{y}^T\right)\mathbf{y}^T/\lambda.$$

Improvements on LESS Solvers

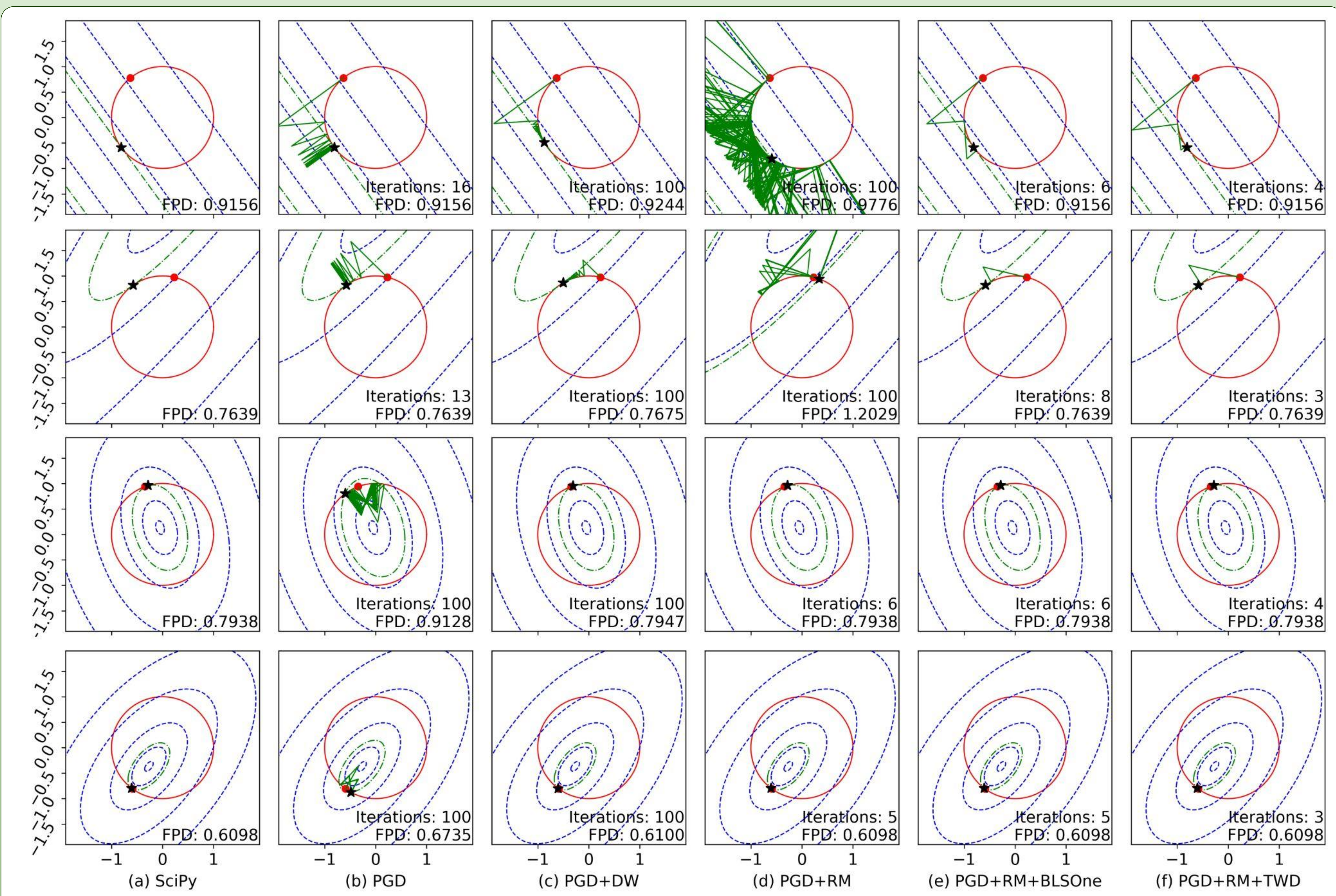


Figure 1. *Iterative optimization in LESS. Each row is a sample with 6 methods. "blue dash": the least squares function; "red solid": the sphere constraints; "green solid": solution updates before and after Riemannian projection; "green dot-dash": the least squares function with the final solution; "red dot": the initial solution; "black star": the final solution, not always the optimal.* ***Ours (the last two columns) require much fewer iterations than the others for the optimal solution with comparable fixed point distance (FPD).***

Table 1. *Effectiveness on 1,000 random Gaussian samples. "In" and "Out": failed cases inner and outer of the sphere respectively. A case is failed when the solution update reaches 100 iterations, "Imp.": the number of cases with energy no greater than SciPy. Problem sizes, m-n for the input matrix, are 2-2 (282 inner and 718 outer), 64-32 (555 inner and 445 outer), and 1024-256 (all inner).*

SciPy	PGD	RM	BLS $\eta=1$	BLS $\eta=\nabla$	TWD	DW	2×2 In↓	2×2 Out↓	2×2 Imp.↑	2×2 MRE↓	64×32 In↓	64×32 Out↓	64×32 Imp.↑	64×32 MRE↓	1024×256 In↓	1024×256 Out↓	1024×256 Imp.↑	1024×256 MRE↓
✓							-	-	-	-	-	-	-	-	-	-	-	-
	✓						172	353	389	10.84	548	440	52	0.15	1000	-	0	1.35
	✓					✓	179	438	37	4.89	474	363	63	0.07	1000	-	0	0.47
	✓	✓					**0**	118	844	4.30	30	**0**	980	**0.00**	4	-	**1000**	**-0.03**
	✓	✓	✓				**0**	14	**938**	0.14	**6**	**0**	**994**	**0.00**	4	-	**1000**	**-0.03**
	✓	✓	✓			✓	175	421	40	3.51	285	142	286	0.01	984	-	999	**-0.03**
	✓	✓		✓			**0**	108	853	4.21	7	**0**	**994**	**0.00**	**2**	-	**1000**	**-0.03**
	✓	✓		✓		✓	204	536	17	1.19	472	355	92	0.04	1000	-	0	0.42
	✓	✓			✓		**0**	**0**	806	**0.00**	29	**0**	954	**0.00**	4	-	**1000**	**-0.03**
	✓	✓			✓	✓	204	533	15	3.12	466	352	90	0.04	1000	-	0	0.42

Improvements on IED Solvers

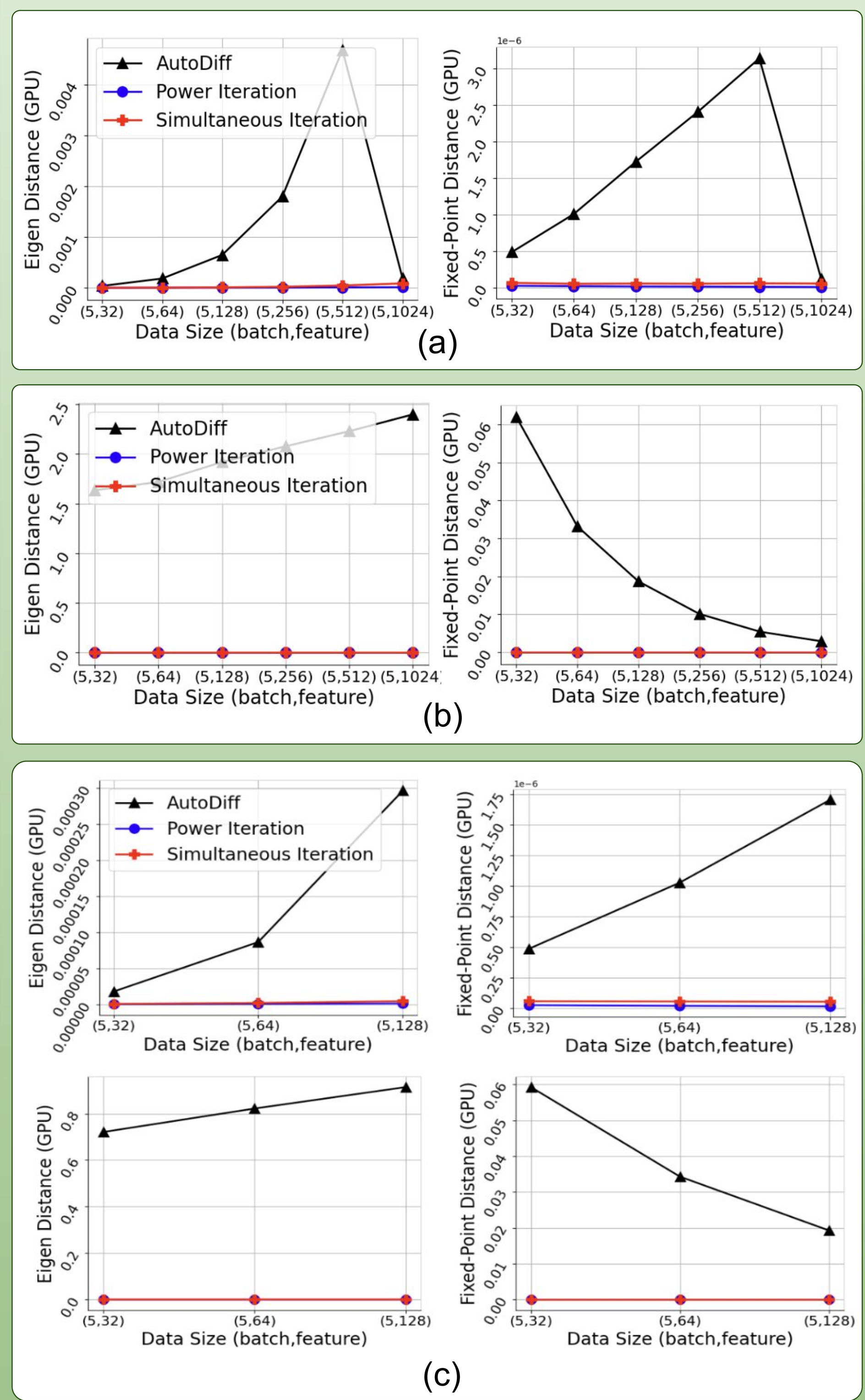


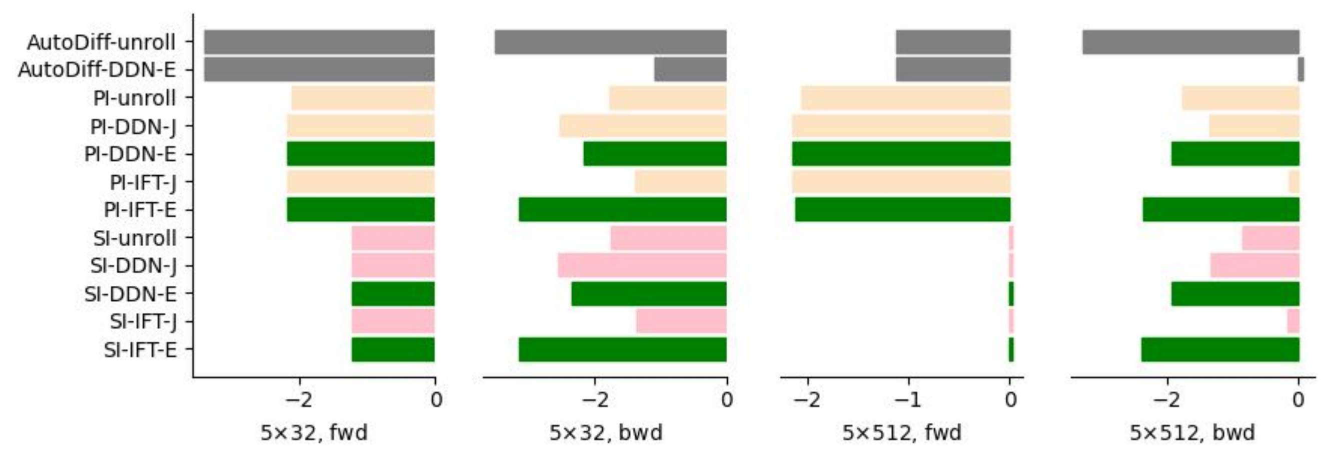
Figure 2. *Accuracy evaluation using eigen distance and FPD (float32). (a) Symmetric and (b) non-symmetric matrices, sampled from the Normal distribution with activation. (c) Matrices from a pretrained ResNet50 with symmetric (top two rows) and non-symmetric (bottom two rows) matrices. Metrics are the less the better.*

Improvements on Backward Efficiency

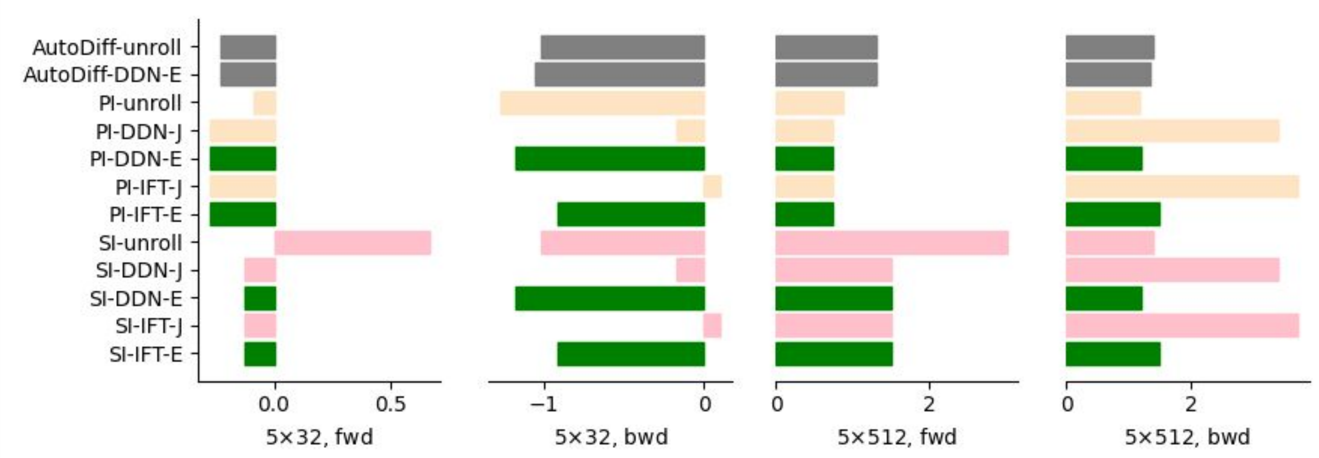
Table 2. *Backward speedup of LESS with exploited structures. Numbers are averaged over 100 samples. "PGD+RM+TWD" is used for LESS. "Speedup" is ("AutoDiff"-"LESS")/"LESS", "AutoDiff" from Scipy solver.*

	Size 256 ×8×64 Time (s)	Size 256 ×8×64 Memory (MB)	Size 256×16×128 Time (s)	Size 256×16×128 Memory (MB)
AutoDiff	4.44	65.22	8.64	519.32
LESS	0.01	49.20	0.02	325.70
Speedup	×444↑	×0.33↑	×432↑	×0.59↑

Figure 3. *IED evaluation on symmetric Gaussian matrices with absolute activation. "fwd": forward pass; "bwd": backward pass; "AutoDiff": PyTorch eigh(); "PI": power iteration; "SI": simultaneous iteration; "unroll": unrolling the forward iteration via PyTorch autodiff mechanism; "J": autodiff Jacobian without exploited structure; "E": ours with exploited structure. Best suggestions are highlighted with green color considering the overall precision in Fig. 2 and computational requirements. More results on non-symmetric matrices and different data types are in the Appendix.*



(a) GPU Time (second). Numbers are rescaled by log10.



(b) GPU Memory (MB). Numbers are rescaled by log10.

Conclusion

- Two deep declarative layers for PMaF, namely LESS and IED, for end-to-end learning
- Overall better and faster (for regularized matrices in IED) solutions using iterative optimization
- Efficient and implicit differentiation with exploited matrix structures
- Code available *https://github.com/anucvml/ddn.git*

References

[1] Gould, S., Hartley, R., and Campbell, D. Deep declarative networks, *TPAMI*, 2022.
[2] Boyd, S. and Vandenberghe, L. Convex optimization. *Cambridge University Press*, 2004.
[3] Gould, S., Campbell, D., Ben-Shabat, I., Koneputugodage, C. H., and Xu, Z. Exploiting problem structure in deep declarative networks: Two case studies, *AAAI Workshop on OTSDM*, 2022.