
Appendix: Compact Memory for Continual Logistic Regression

Anonymous Author(s)

Affiliation

Address

email

1 Contents

2	A Appendix: Methodology details	2
3	A.1 Background: PPCA and its EM algorithm	2
4	A.2 Details for case of Multi-label classification	4
5	B Appendix: Experiment details	5
6	B.1 Multi-output regression on Split-MNIST	5
7	B.2 Binary logistic regression on four-moon dataset	6
8	B.3 Binary logistic regression on USPS odd vs even dataset	7
9	B.4 Multi-label logistic regression on Split CIFAR-10, CIFAR-100, and TinyImageNet	8
10	B.5 Additional results on Permuted MNIST	9

11 A Appendix: Methodology details

12 A.1 Background: PPCA and its EM algorithm

13 To aid in understanding our algorithm for learning the compact memory, we briefly introduce
 14 Probabilistic PCA (PPCA) and the Expectation-Maximization (EM) algorithm in our context; further
 15 details can be checked in [Tipping and Bishop, 1999a,b]. To this end, we first remind that our compact
 16 memory $\mathbf{U}_{t+1} \in \mathbb{R}^{H \times K_{t+1}}$ serves as the weight parameter in the following linear model with latent
 17 variable \mathbf{Z}_{t+1} ,

$$[\mathbf{U}_t, \mathbf{\Phi}_{t+1}] \approx \mathbf{U}_{t+1} \mathbf{Z}_{t+1}, \quad (1)$$

18 where the feature matrix $\mathbf{\Phi}_{t+1} \in \mathbb{R}^{H \times N_{t+1}}$ and the previous memory $\mathbf{U}_t \in \mathbb{R}^{H \times K_t}$ are assumed to
 19 be given observations.

20 From now, for brevity, we omit subscript and notate $N := K_{t+1} + N_{t+1}$, $\mathbf{U} := \mathbf{U}_{t+1} \in \mathbb{R}^{H \times K_{t+1}}$,
 21 $\mathbf{\Phi} := [\mathbf{U}_t, \mathbf{\Phi}_{t+1}] \in \mathbb{R}^{H \times (N_{t+1} + K_t)}$, where $\phi_n \in \mathbb{R}^H$ denotes n -th column of $\mathbf{\Phi}$.

22 **PPCA Model.** With Gaussian latent variable $\mathbf{z}_n \sim \mathcal{N}(\mathbf{0}_H, \mathbf{I}_{H \times H})$, the n -th observation ϕ_n can
 23 be modeled as

$$\phi_n = \mathbf{U} \mathbf{z}_n + \boldsymbol{\eta},$$

24 where $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}_H, \epsilon \mathbf{I}_{H \times H})$ denotes the Gaussian noise with the scalar hyperparameter $\epsilon \in \mathbb{R}_+$. Then,
 25 for $\mathbf{u} := 1/N \sum_n \phi_n$ and $\mathbf{C} := \mathbf{U} \mathbf{U}^\top + \epsilon \mathbf{I}$, the marginal likelihood of ϕ_n is derived as

$$\begin{aligned} p(\phi_n) &= \int \underbrace{p(\phi_n | \mathbf{z}_n)}_{\phi_n \sim \mathcal{N}(\mathbf{U} \mathbf{z}_n + \mathbf{u}, \epsilon \mathbf{I}_{H \times H})} \underbrace{p(\mathbf{z}_n)}_{\mathbf{z}_n \sim \mathcal{N}(\mathbf{0}_H, \mathbf{I}_{H \times H})} d\mathbf{z}_n \\ &= (2\pi)^{-H/2} |\mathbf{C}| \exp \left(-\frac{1}{2} (\phi_n - \mathbf{u})^\top \mathbf{C}^{-1} (\phi_n - \mathbf{u}) \right). \end{aligned} \quad (2)$$

26 Thus, the log likelihood of Eq.(1) is written as

$$\log p(\mathbf{\Phi}) = \sum_{n=1}^N \log p(\phi_n) = N \left((H/2) \log 2\pi + \text{tr}(\mathbf{C}^{-1} \mathbf{S}) + \log |\mathbf{C}| \right), \quad (3)$$

27 where $\mathbf{S} = (1/N) \sum_n (\phi_n - \mathbf{u})(\phi_n - \mathbf{u})^\top$.

28 **Stationary condition.** Because the gradient of the log marginal likelihood $\nabla_{\mathbf{U}} \log p(\mathbf{\Phi})$ is given as

$$\nabla_{\mathbf{U}} \log p(\mathbf{\Phi}) = N (\mathbf{C}^{-1} \mathbf{S} \mathbf{C}^{-1} \mathbf{U} - \mathbf{C}^{-1} \mathbf{U}), \quad (4)$$

29 the optimal \mathbf{U} maximizing the $\log p(\mathbf{\Phi})$, satisfies the following equation

$$\mathbf{S} \mathbf{C}^{-1} \mathbf{U} = \mathbf{U}, \quad (5)$$

30 due to the stationary condition $\nabla_{\mathbf{U}} \log p(\mathbf{\Phi}) = \mathbf{0}$.

31 Eq.(5) implies that if (i) $\mathbf{U} \mathbf{U}^\top$ is invertible, meaning that \mathbf{U} is a full rank matrix with rank H and (ii)
 32 $\mathbf{u} = \mathbf{0}_H$, meaning that the columns of $\mathbf{\Phi}$ have zero mean, then the following holds

$$\mathbf{S} = \mathbf{C} \quad \Leftrightarrow \quad (1/N) \mathbf{\Phi} \mathbf{\Phi}^\top = \mathbf{U} \mathbf{U}^\top + \epsilon \mathbf{I}, \quad (6)$$

33 where the right side is equal to the hessian matching objective for linear regression up to constant N .

34 **Motivation for Hessian matching via PPCA.** Eq.(6) suggests that if we find the optimal weight \mathbf{U}
 35 in PPCA formulation, we can find the memory for Hessian matching objective in our problem.

36 **EM algorithm.** Finding the optimal weight \mathbf{U} in PPCA can be conducted by EM algorithm for
 37 computational efficiency.

38 The EM algorithm maximizes $\ell(\mathbf{U})$ with respect to \mathbf{U} in order to find the stationary point $\mathbf{S} = \mathbf{C}$ in
 39 Eq.(6) where $\ell(\mathbf{U})$ appears in the lower bound of the marginal likelihood $\log p(\Phi)$, as follow:

$$\log p(\Phi) = \log \int \left(p(\Phi|\mathbf{Z}) \frac{p(\mathbf{Z})}{p(\mathbf{Z}|\mathbf{U}, \Phi)} \right) p(\mathbf{Z}|\mathbf{U}, \Phi) d\mathbf{Z} \geq \underbrace{\int \log p(\Phi, \mathbf{Z}) p(\mathbf{Z}|\mathbf{U}, \Phi) d\mathbf{Z}}_{:=\ell(\mathbf{U})} + H(p(\mathbf{Z}|\mathbf{U}, \Phi)).$$

40 The EM algorithm consists of the following alternative update procedure:

41 **E-step:** Update posterior of latent variable $p(\mathbf{Z}|\mathbf{U}^{(\text{old})}, \Phi)$ and then compute the expected loss

$$\ell(\mathbf{U}; \mathbf{U}^{(\text{old})}) := \int \log p(\Phi, \mathbf{Z}) p(\mathbf{Z}|\mathbf{U}^{(\text{old})}, \Phi) d\mathbf{Z},$$

42 where $\mathbf{U}^{(\text{old})}$ denotes the weight updated in previous iteration. With $\mathbf{M} := \mathbf{U}^{(\text{old})\top} \mathbf{U}^{(\text{old})} + \epsilon \mathbf{I} \in$
 43 $\mathbb{R}^{K_{t+1} \times K_{t+1}}$, the posterior of latent variable for each \mathbf{z}_n is computed as

$$p(\mathbf{z}_n|\mathbf{U}^{(\text{old})}, \Phi) = \mathcal{N}(\mathbf{M}^{-1} \mathbf{U}^{(\text{old})\top} (\phi_n - \mathbf{u}), \epsilon \mathbf{M}^{-1}).$$

44 Thus, the expected incomplete likelihood $\ell(\mathbf{U}; \mathbf{U}^{(\text{old})})$ is computed as

$$\begin{aligned} \ell(\mathbf{U}; \mathbf{U}^{(\text{old})}) &= \sum_{n=1}^N \int \log p(\phi_n, \mathbf{z}_n) p(\mathbf{z}_n|\mathbf{U}^{(\text{old})}, \Phi) d\mathbf{z}_n \\ &= \sum_{n=1}^N \left(\frac{H}{2} \log \epsilon + \frac{1}{2} \text{tr}(\mathbf{E}[\mathbf{z}_n \mathbf{z}_n^\top]) + \frac{1}{2\epsilon} \|\phi_n - \mathbf{u}\|^2 - \frac{1}{\epsilon} \mathbf{E}[\mathbf{z}_n^\top] \mathbf{U}^\top (\phi_n - \mathbf{u}) + \frac{1}{2\epsilon} \text{tr}(\mathbf{U}^\top \mathbf{U} \mathbf{E}[\mathbf{z}_n \mathbf{z}_n^\top]) \right), \end{aligned}$$

45 where $\mathbf{E}[\mathbf{z}_n] = \mathbf{M}^{-1} \mathbf{U}^{(\text{old})\top} (\phi_n - \mathbf{u})$ and $\mathbf{E}[\mathbf{z}_n \mathbf{z}_n^\top] = \epsilon \mathbf{M}^{-1} + \mathbf{E}[\mathbf{z}_n] \mathbf{E}[\mathbf{z}_n]^\top$.

46 **M-step:** Update the parameter $\mathbf{U}^{(\text{old})}$ newly by

$$\mathbf{U}^{(\text{old})} \leftarrow \arg \max_{\mathbf{U}} \ell(\mathbf{U}; \mathbf{U}^{(\text{old})}),$$

47 which is given as

$$\mathbf{U}^{(\text{old})} = \left(\sum_{n=1}^N (\phi_n - \mathbf{u}) \mathbf{E}[\mathbf{z}_n]^\top \right) \left(\sum_{n=1}^N \mathbf{E}[\mathbf{z}_n \mathbf{z}_n^\top] \right)^{-1}.$$

48 We repeat **E-step** and **M-step** alternatively until the learnable parameter \mathbf{U} converges the local
 49 optimum.

50 A.2 Details for case of Multi-label classification

51 **Understanding on K-prior usign Hessian.** Let us remind the model parameter $\Theta =$
 52 $[\theta_1 \dots \theta_C]^\top \in R^{C \times H}$ for C -class classification and its Hessian $\nabla^2 \ell(\mathbf{f}, \mathbf{y}) = \sigma'(\mathbf{f}) \otimes \phi \phi^\top \in$
 53 $R^{CH \times CH}$ with $\mathbf{f} = [\mathbf{f}^{(1)} \dots \mathbf{f}^{(C)}]$ and its softmax output $\sigma(\mathbf{f}) = [\sigma^{(1)}(\mathbf{f}) \dots \sigma^{(C)}(\mathbf{f})] \in \Delta^{C-1}$.
 54 Then, the derivative of softmax output is given as $[\sigma'(\mathbf{f})]_{ij} = \sigma^{(i)}(\mathbf{f})(1 - \sigma^{(i)}(\mathbf{f}))$ if $i = j$, otherwise
 55 $[\sigma'(\mathbf{f})]_{ij} = -\sigma^{(i)}(\mathbf{f})\sigma^{(j)}(\mathbf{f})$.
 56 For multi-label classification task, the K-prior regularization $\mathcal{K}_t(\theta)$ at t -task model parameter Θ_t ,
 57 can be approximated with the second-order Taylor approximation, as follows:

$$\begin{aligned}
 \mathcal{K}_t(\theta) &\approx \sum_{i \in \mathcal{D}_t} \text{vec}(\Theta - \Theta_t)^\top \left(\sigma'(\mathbf{f}_i) \otimes \phi_i \phi_i^\top \right) \text{vec}(\Theta - \Theta_t) + \delta \text{vec}(\Theta - \Theta_t)^\top \text{vec}(\Theta - \Theta_t) \\
 &= \sum_{i \in \mathcal{D}_t} \sum_{c=1}^C (\theta^{(c)} - \theta_t^{(c)})^\top \left(\sigma^{(c)}(\mathbf{f}_i)(1 - \sigma^{(c)}(\mathbf{f}_i)) \phi_i \phi_i^\top \right) (\theta^{(c)} - \theta_t^{(c)}) \\
 &\quad - 2 \sum_{i \in \mathcal{D}_t} \sum_{c_1 < c_2} \underbrace{(\theta^{(c_1)} - \theta_t^{(c_1)})^\top (\sigma^{(c_1)}(\mathbf{f}_i) \sigma^{(c_2)}(\mathbf{f}_i) \phi_i \phi_i^\top) (\theta^{(c_2)} - \theta_t^{(c_2)})}_{\sigma^{(c_1)}(\mathbf{f}_i) \sigma^{(c_2)}(\mathbf{f}_i) \approx 0 \text{ in most cases}} + \delta \text{vec}(\Theta - \Theta_t)^\top \text{vec}(\Theta - \Theta_t) \\
 &\approx \sum_{i \in \mathcal{D}_t} \sum_{c=1}^C (\theta^{(c)} - \theta_t^{(c)})^\top \left(\sigma^{(c)}(\mathbf{f}_i)(1 - \sigma^{(c)}(\mathbf{f}_i)) \phi_i \phi_i^\top \right) (\theta^{(c)} - \theta_t^{(c)}) + \delta \text{vec}(\Theta - \Theta_t)^\top \text{vec}(\Theta - \Theta_t) \\
 &= \sum_{c=1}^C (\theta^{(c)} - \theta_t^{(c)})^\top \left(\underbrace{\sum_{i \in \mathcal{D}_t} \sigma^{(c)}(\mathbf{f}_i)(1 - \sigma^{(c)}(\mathbf{f}_i)) \phi_i \phi_i^\top}_{\text{Hessian of binary classification for } c\text{-class}} + \delta \mathbf{I}_{H \times H} \right) (\theta^{(c)} - \theta_t^{(c)}),
 \end{aligned}$$

58 where the second approximation is considered because Θ_t is obtained after learning task \mathcal{D}_t and
 59 thus $\sigma(\mathbf{f})$ would have a peak predictive probability for some class $c \in [C]$ in most cases, yielding
 60 $\sigma(\mathbf{f}_i^{(c_1)}) \sigma(\mathbf{f}_i^{(c_2)}) \approx 0$. The above approximation, resulting

$$\mathcal{K}_t(\theta) \approx \sum_{c=1}^C (\theta^{(c)} - \theta_t^{(c)})^\top \left(\underbrace{\sum_{i \in \mathcal{D}_t} \sigma^{(c)}(\mathbf{f}_i)(1 - \sigma^{(c)}(\mathbf{f}_i)) \phi_i \phi_i^\top}_{\text{Hessian of binary classification for } c\text{-class}} + \delta \mathbf{I}_{H \times H} \right) (\theta^{(c)} - \theta_t^{(c)}),$$

61 implies that for given $\phi_i \in \mathcal{D}_t$, if there is some $c \in [C]$ satisfying $\sigma^{(c)}(\mathbf{f}_i)(1 - \sigma^{(c)}(\mathbf{f}_i))$ to be large
 62 value, then compact memory should learn ϕ_i for the regularization of c -class parameter $\theta^{(c)}$.

63 **Motivation for Hessian matching.** This encourages us to learn compact memory $\{(w_{k|t}, \mathbf{u}_{k|t})\}_{k=1}^{K_t}$
 64 satisfying

$$\sum_{i \in \mathcal{D}_t} \underbrace{\left(\sum_{c=1}^C \sigma^{(c)}(\mathbf{f}_i)(1 - \sigma^{(c)}(\mathbf{f}_i)) \right)}_{\text{importance for } \phi_i \text{ over } C \text{ classes}} \phi_i \phi_i^\top \approx \sum_k w_{k|t} \left(\sum_{c=1}^C \sigma^{(c)}(\mathbf{f}_{\mathbf{u}_{k|t}})(1 - \sigma^{(c)}(\mathbf{f}_{\mathbf{u}_{k|t}})) \right) \mathbf{u}_{k|t} \mathbf{u}_{k|t}^\top,$$

65 which yields the generalization of hessian matching for multi-label classification.

66 **Remark.** If $\sigma(\mathbf{f})$ is a peaked probability meaning that there is some $c \in [C]$ satisfying $\sigma^{(c)}(\mathbf{f}) \approx 1$,
 67 then $\sigma^{(i)}(\mathbf{f})(1 - \sigma^{(i)}(\mathbf{f})) = 0$ for all class $i \in [C]$. Thus, for all features of \mathcal{D}_t , if their $\{\sigma(\mathbf{f})\}$ have a
 68 peaked probability, there is no information to learn memory. Therefore, for small $\zeta > 0$ we consider
 69 the clipping for logit is necessary, i.e., $\max_i \sigma^{(i)}(\mathbf{f}) \leq 1 - \zeta$ and $\min_i \sigma^{(i)}(\mathbf{f}) \geq \zeta$. In this work, we
 70 use $\zeta = 10^{-4}$.

B Appendix: Experiment details

Reproducibility. Our implementation and additional results are available at https://anonymous.4open.science/r/project_cptmem-FFB3.

Computation resource. We use RTX 6000 Ada for experiments on Permuted-MNIST and Split-TinyImageNet. For other experiments, we use RTX-3090.

B.1 Multi-output regression on Split-MNIST

Experiment setting. We consider the following hyperparameters:

- For feature map $\phi(\mathbf{x})$, we use the identity map where the raw pixel values of the image serve directly as features.
- For hyperparameters of learning model parameter θ_t , we use Adam optimizer with learning $\text{lr} = 10^{-3}$. We use 100 epochs for each task. For the weight-space regularization hyperparameter δ , we use $\delta = 0.01$.
- For hyperparameters of learning memory \mathbf{U}_{t+1} , we run 10,000 iterations for each task. For the noise hyperparameter ϵ for the EM algorithm, we use $\epsilon = 10^{-3}$.

Additional results. Table 1 compares the performances of our method and the SVD approach, which is regarded as the oracle method. For memory size K_{t+1} , the SVD approach obtains the optimal memory

$$\mathbf{U}_{t+1} = \mathbf{U}_{1:K_{t+1}} \text{diag}(\mathbf{d}_{1:K_{t+1}}^{1/2})$$

sequentially where the columns of eigenvectors $\mathbf{U} \in \mathbb{R}^{H \times R}$ with its rank R and the corresponding eigenvalues $\mathbf{d} \in \mathbb{R}_+^R$ are given by $\text{eigh}(\Phi_{t+1} \Phi_{t+1}^\top + \mathbf{U}_t \mathbf{U}_t^\top) = \mathbf{U} \text{diag}(\mathbf{d}) \mathbf{U}^\top$ for the feature matrix $\Phi_{t+1} \in \mathbb{R}^{H \times N_{t+1}}$ and the previous memory $\mathbf{U}_t \in \mathbb{R}^{H \times K_t}$. The ΔMem denotes the memory size allocated per task; for example, if $\Delta \text{Mem} = 10$ is used, the memory increases by 10 for each task.

This result demonstrates that our method is effective in mitigating catastrophic forgetting, especially when using a small amount of memory because the performances on previous tasks, achieved by our method, tend to be superior to that of the SVD approach.

Table 1: Multi-output regression on Split-MNIST over 3 runs. The performance of batch training is obtained as 0.839, comparable to the **Avg** of each method.

ΔMem	Method	Task 1	Task 2	Task 3	Task 4	Task 5	Avg
10	OUR	0.920 \pm 0.001	0.474 \pm 0.012	0.616 \pm 0.003	0.602 \pm 0.001	0.874 \pm 0.001	0.697 \pm 0.004
	SVD	0.776 \pm 0.011	0.337 \pm 0.005	0.592 \pm 0.006	0.587 \pm 0.004	0.905 \pm 0.002	0.639 \pm 0.002
20	OUR	0.941 \pm 0.000	0.566 \pm 0.001	0.695 \pm 0.002	0.680 \pm 0.003	0.837 \pm 0.000	0.744 \pm 0.001
	SVD	0.831 \pm 0.010	0.417 \pm 0.006	0.652 \pm 0.004	0.641 \pm 0.002	0.884 \pm 0.004	0.685 \pm 0.003
40	OUR	0.959 \pm 0.000	0.689 \pm 0.001	0.740 \pm 0.001	0.786 \pm 0.001	0.816 \pm 0.001	0.798 \pm 0.000
	SVD	0.877 \pm 0.006	0.535 \pm 0.003	0.688 \pm 0.004	0.735 \pm 0.004	0.889 \pm 0.001	0.745 \pm 0.001
160	OUR	0.966 \pm 0.000	0.764 \pm 0.002	0.738 \pm 0.001	0.848 \pm 0.001	0.785 \pm 0.000	0.820 \pm 0.000
	SVD	0.895 \pm 0.006	0.687 \pm 0.006	0.744 \pm 0.002	0.862 \pm 0.003	0.886 \pm 0.001	0.815 \pm 0.002
320	OUR	0.965 \pm 0.000	0.775 \pm 0.002	0.720 \pm 0.001	0.842 \pm 0.001	0.773 \pm 0.001	0.815 \pm 0.001
	SVD	0.901 \pm 0.006	0.730 \pm 0.005	0.751 \pm 0.004	0.874 \pm 0.003	0.887 \pm 0.002	0.828 \pm 0.002

95 B.2 Binary logistic regression on four-moon dataset

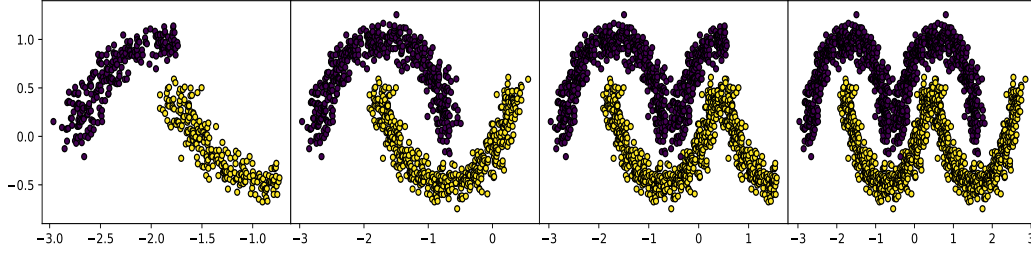


Figure 1: Sequences of four-moon classification task

96 **Experiment setting.** We split all data points of Four-moon datasets into 4 tasks of
 97 $[500, 500, 500, 500]$ where the task proceeds as shown in Figure 1.

98 We consider the following hyperparameters:

- 99 • For feature map $\phi(\mathbf{x})$, we use polynomial feature $\text{poly}(5)$ yielding 21 feature dimension. We use
 100 the scikit-learn package.
- 101 • For hyperparameters of learning model parameter θ_t , we use Adam optimizer with learning
 102 $\text{lr} = 0.01$. We set the number of iterations across $\{10000, 20000\}$ to learn the model parameter θ_t
 103 for each task; we confirm the convergence of the training loss. For weight-space regularization
 104 hyperparameter δ , we set $\delta = 10^{-2}$.
- 105 • For hyperparameters of learning memory \mathbf{U}_{t+1} , we set the number of iterations across
 106 $\{50, 100, 200\}$ for each task. For the memory size, we consider $\{7, 14, 21\}$ memory for each
 107 task and accumulate the memory as the task proceeds. For the noise hyperparameter ϵ , we conduct
 108 a grid search over $\epsilon \in \{10^{-3}, 10^{-4}, 10^{-5}\}$ and use 10^{-4} .

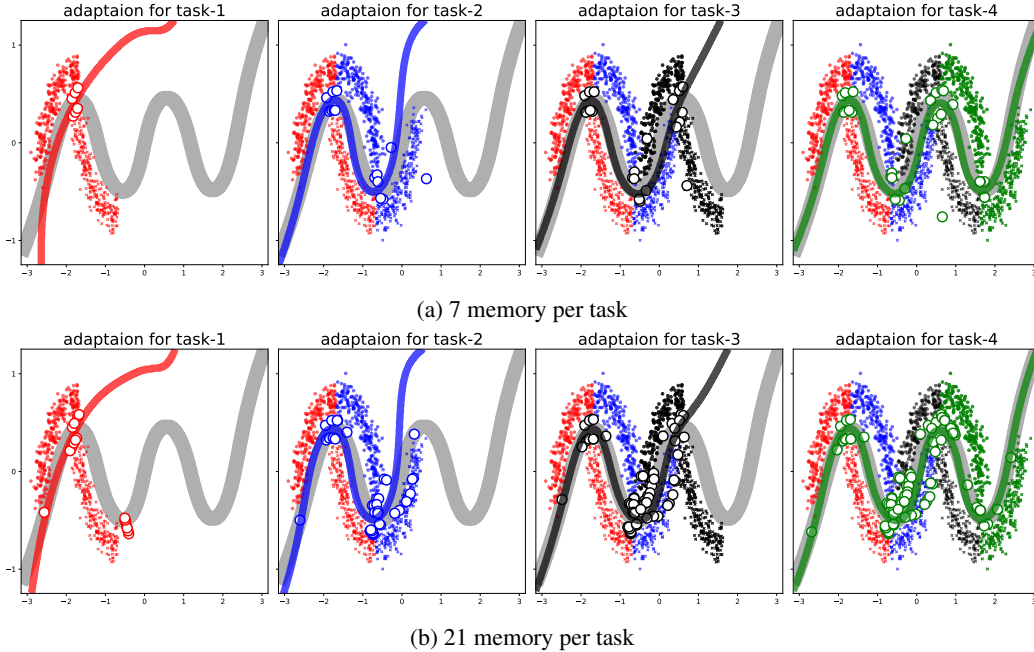


Figure 2: Investigation on varying memory size used for K-prior.

109 **Additional results across varying memory size.** Figure 2 shows the results using the 7 and 21
 110 memory per task. The result using 14 memory is reported in Section 4 of the main manuscript. This
 111 confirms that our method can match the decision boundary of batch training closely even when using
 112 a small amount of memory, as shown in Figure 2a. As more memory is used, our decision boundary
 113 becomes more equal to that of the batch training, as shown in Figure 2b.

114 B.3 Binary logistic regression on USPS odd vs even dataset

115 **Experiment setting.** We consider the following hyperparameters:

- 116 • For feature map $\phi(\mathbf{x})$, we use polynomial feature $\text{poly}(1)$ yielding 256 feature dimension. We use
117 the scikit-learn package.
- 118 • For hyperparameters of learning model parameter θ_t , we use Adam optimizer with learning
119 $\text{lr} = 0.1$. We use 5000 iterations to learn the model parameter θ_t for each task. For the weight-
120 space regularization hyperparameter δ , we conduct a grid search over $\delta \in \{0.5, 0.1, 0.05\}$ and use
121 $\delta = 0.5$ for K-prior and our method.
- 122 • For hyperparameters of learning memory \mathbf{U}_{t+1} , we use 5000 iterations for each task. For the noise
123 hyperparameter ϵ , we conduct a grid search over $\epsilon \in \{10^{-3}, 10^{-4}, 10^{-5}\}$.

124 **Investigation on varying weight-space regularization hyperparameter for K-prior.** Figure 3
125 compares K-prior and our method across the varying hyperparameters $\delta \in \{0.5, 0.1, 0.05\}$ that control
126 the importance of weight-space regularization hyperparameter term for K-prior. This result confirms
127 that our method improves the memory efficiency in general and is more effective at $\delta \in \{0.1, 0.05\}$
128 where the K-prior can match the result of batch training more closely.

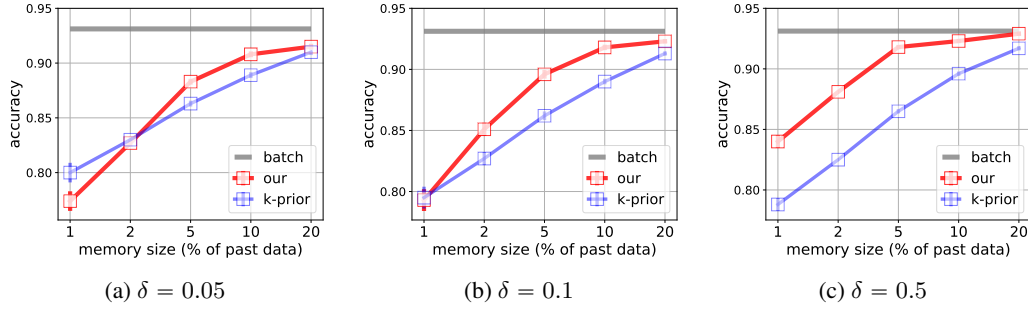


Figure 3: Investigation on varying hyperparameter $\delta \in \{0.05, 0.1, 0.5\}$ for K-prior

129 **Investigation on varying noise hyperparameter EM algorithm for PPCA.** Table 2 compares the
130 performances of our method across varying noise hyperparameter ϵ in the EM algorithm for PPCA
131 where we run 5 experiments with different random seeds. This confirms that our method is consistent
over the noise hyperparameter ϵ .

Table 2: Investigation on the effect of noise hyperparameter $\epsilon \in \{10^{-3}, 10^{-4}, 10^{-5}\}$.

ΔMem	ϵ	Task 1	Task 2	Task 3	Task 4	Task 5	Avg
1%	$\epsilon = 10^{-3}$	0.769 ± 0.046	0.840 ± 0.012	0.566 ± 0.033	0.982 ± 0.005	0.946 ± 0.002	0.820 ± 0.013
	$\epsilon = 10^{-4}$	0.791 ± 0.029	0.855 ± 0.014	0.623 ± 0.022	0.983 ± 0.003	0.952 ± 0.005	0.840 ± 0.007
	$\epsilon = 10^{-5}$	0.772 ± 0.043	0.872 ± 0.010	0.627 ± 0.032	0.980 ± 0.003	0.954 ± 0.002	0.841 ± 0.008
2%	$\epsilon = 10^{-3}$	0.860 ± 0.010	0.913 ± 0.002	0.689 ± 0.020	0.979 ± 0.003	0.934 ± 0.003	0.875 ± 0.003
	$\epsilon = 10^{-4}$	0.885 ± 0.012	0.885 ± 0.018	0.724 ± 0.008	0.981 ± 0.003	0.933 ± 0.005	0.881 ± 0.004
	$\epsilon = 10^{-5}$	0.891 ± 0.006	0.893 ± 0.012	0.715 ± 0.010	0.982 ± 0.003	0.937 ± 0.002	0.883 ± 0.003
5%	$\epsilon = 10^{-3}$	0.932 ± 0.002	0.931 ± 0.003	0.810 ± 0.011	0.980 ± 0.003	0.920 ± 0.003	0.915 ± 0.002
	$\epsilon = 10^{-4}$	0.950 ± 0.002	0.921 ± 0.003	0.819 ± 0.012	0.983 ± 0.004	0.915 ± 0.005	0.918 ± 0.002
	$\epsilon = 10^{-5}$	0.948 ± 0.002	0.921 ± 0.007	0.798 ± 0.009	0.984 ± 0.002	0.915 ± 0.004	0.913 ± 0.002
10%	$\epsilon = 10^{-3}$	0.933 ± 0.002	0.944 ± 0.005	0.876 ± 0.008	0.972 ± 0.004	0.878 ± 0.003	0.921 ± 0.002
	$\epsilon = 10^{-4}$	0.962 ± 0.002	0.937 ± 0.003	0.855 ± 0.009	0.976 ± 0.003	0.884 ± 0.003	0.923 ± 0.002
	$\epsilon = 10^{-5}$	0.969 ± 0.003	0.934 ± 0.002	0.855 ± 0.009	0.977 ± 0.003	0.889 ± 0.003	0.925 ± 0.001
20%	$\epsilon = 10^{-3}$	0.949 ± 0.003	0.946 ± 0.002	0.891 ± 0.005	0.971 ± 0.002	0.859 ± 0.002	0.923 ± 0.001
	$\epsilon = 10^{-4}$	0.979 ± 0.001	0.949 ± 0.002	0.877 ± 0.006	0.965 ± 0.002	0.872 ± 0.003	0.929 ± 0.002
	$\epsilon = 10^{-5}$	0.981 ± 0.001	0.940 ± 0.001	0.868 ± 0.006	0.967 ± 0.002	0.876 ± 0.004	0.927 ± 0.002

B.4 Multi-label logistic regression on Split CIFAR-10, CIFAR-100, and TinyImageNet

Experiment setting. We consider the following hyperparameters:

- For feature map $\phi(\mathbf{x})$, we use the feature extractor of ResNet-50 and Vision transformer (Vit) that are pretrained by CLIP. For Split CIFAR-10, we use Vit-B/32 meaning the base-scale model with 32 patch size. For Split CIFAR-100 and Split Tiny-ImageNet, we use Vit-L/14 meaning the large-scale model size with 14 patch size. The feature extractors of ResNet-50 and Vit yield 2048 features and 768 features, respectively.
- For hyperparameters of learning model parameter θ_t , we use Adam optimizer with learning $\text{lr} = 0.1$. We use 100 iterations for each task with 1024 batch size. For the weight-space regularization hyperparameter δ , we conduct a grid search over $\delta \in \{0.1, 0.01, 0.001\}$.
- For hyperparameters of learning memory \mathbf{U}_{t+1} , we set 10 iterations for each task. For the noise hyperparameter ϵ , we use $\epsilon = 10^{-4}$.

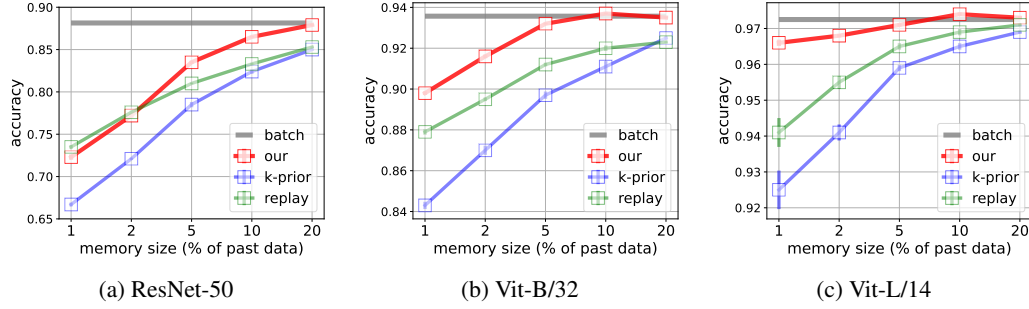


Figure 4: Investigation on varying feature extractor with Split-CIFAR-10.

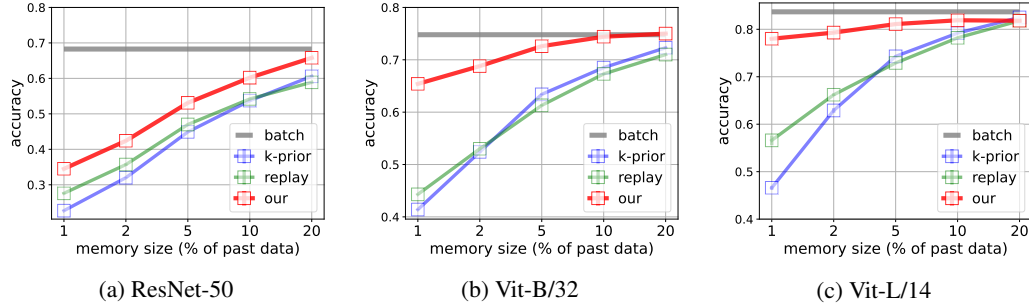


Figure 5: Investigation on varying feature extractor with Split-CIFAR-100.

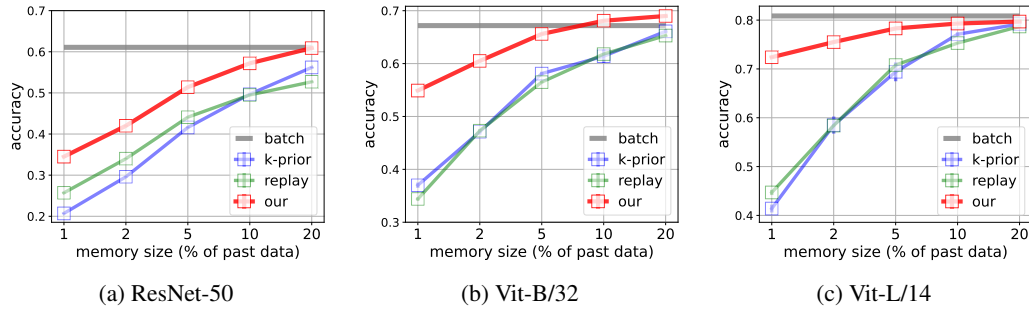


Figure 6: Investigation on varying feature extractor with Split-TinyImageNet.

Investigation on varying feature extractor. Figure 4 shows the results on Split-CIFAR 10 across varying feature extractors where the feature extractors of ResNet-50, Vit-B/32, and Vit-L/14 are used in Figures 4a to 4c, respectively. Similarly, Figures 5 and 6 show the corresponding results on Split-CIFAR-100 and Split-TinyImageNet, respectively. These results confirm that our method improves memory efficiency consistently regardless of the feature extractor.

B.5 Additional results on Permuted MNIST

Experiment setting. Following the benchmark experiment setting in [Lomonaco et al., 2021], we also consider Permuted-MNIST consisting of a sequence of 5 tasks constructed by permuting pixels of 60,000 training samples of MNIST for each task.

We consider the following hyperparameters:

- For feature map $\phi(\mathbf{x})$, we use the feature extractor of ResNet-50 and Vision transformer (Vit) that are pretrained by CLIP. The feature extractor of ResNet-50 and Vit yield 2048 features and 768 features, respectively
- For hyperparameters of learning model parameter θ_t , we use Adam optimizer with learning $\text{lr} = 0.1$. We use 200 iterations for each task. For the weight-space regularization hyperparameter δ , we conduct a grid search over $\delta \in \{0.01, 0.001\}$.
- For hyperparameters of learning memory \mathbf{U}_{t+1} , we set 10 iterations for each task. For the noise hyperparameter ϵ , we consider $\epsilon = 10^{-4}$.

Additional results. Table 3 shows that our method outperforms other baselines especially when 1, 2, and, 5 percent of data points are used as memory. This result confirms that our method also improves memory efficiency on domain incremental task .

Table 3: Performance on Permuted-MNIST across varying amount of memory.

Feature	Method	1%	2%	5%	10%
ResNet-50	Replay	0.474 ± 0.002	0.508 ± 0.002	0.554 ± 0.001	$\mathbf{0.592} \pm 0.002$
	K-prior	0.465 ± 0.003	0.493 ± 0.002	0.534 ± 0.002	0.572 ± 0.003
	Our	$\mathbf{0.490} \pm 0.001$	$\mathbf{0.521} \pm 0.001$	$\mathbf{0.561} \pm 0.002$	$\mathbf{0.592} \pm 0.003$
Vit-B/32	Replay	0.490 ± 0.003	0.524 ± 0.002	0.578 ± 0.004	$\mathbf{0.623} \pm 0.003$
	K-prior	0.453 ± 0.001	0.489 ± 0.003	0.549 ± 0.005	0.594 ± 0.004
	Our	$\mathbf{0.522} \pm 0.006$	$\mathbf{0.545} \pm 0.007$	$\mathbf{0.584} \pm 0.003$	0.613 ± 0.004

References

- Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 61(3):611–622, 1999a.
- Michael E Tipping and Christopher M Bishop. Mixtures of probabilistic principal component analyzers. *Neural computation*, 11(2):443–482, 1999b.
- Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas Tolas, Simone Scardapane, Luca Antiga, Subutai Amhad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide Bacciu, and Davide Maltoni. Avalanche: an end-to-end library for continual learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2nd Continual Learning in Computer Vision Workshop, 2021.