An Optimal Transport-driven Approach for Cultivating Latent Space in Online Incremental Learning

Supplementary Material

7. Background

7.1. Gaussian Mixture Model

Formally, a Gaussian Mixture Model (GMM) is a probability distribution composed of several Gaussian components. For a given number of components K, it can be expressed as:

$$\sum_{k=1}^{K} \pi_k \mathcal{N}_k,$$

where each $\mathcal{N}_k$ denotes a Gaussian distribution and the weights satisfy $\sum_{k=1}^{K} \pi_k = 1$. We denote by $GMM_d(K)$ the subset of probability measures in $\mathbb{R}^d$ that can be represented as a Gaussian mixture with at most K components. Note that K can potentially be infinite. The GMM setting is not only a fundamental object in statistical problems but also finds numerous applications, such as image segmentation [21], anomaly detection [79], keystroke recognition [30].

Given n i.i.d samples from a distribution $\mathcal{P} \in GMM_d(K)$, the parameters of the GMM representation of $\mathcal{P}$ are typically estimated via maximum likelihood using the Expectation-Maximization (EM) algorithm [19] with the computational complexity of cubic order (i.e $\mathcal{O}(nKd^3)$). Later, the approach of [50] lowers the computational complexity to $\mathcal{O}(nKd^2)$ by formulating expressions based on precision matrices in place of covariance matrices.

The connection between Wasserstein metrics and Gaussian Mixture Models is initially established via the derivation of the optimal transport problem between two GMMs $\sum_{i=1}^{K_1} \pi_i^{(\mathcal{N})} \mathcal{N}_i$ and $\sum_{j=1}^{K_2} \pi_j^{(\mathcal{P})} \mathcal{P}_j$ which can be formulated as the following optimization problem [18]:

$$\min_{\gamma \in \Gamma\left(\pi^{(\mathcal{N})}, \pi^{(\mathcal{P})}\right)} \sum_{i=1}^{K_1} \sum_{j=1}^{K_2} \gamma_{i,j} \mathcal{W}_2(\mathcal{N}_i, \mathcal{P}_j).$$

Moreover, [78] applies Wasserstein Gradient Flows to GMM policy optimization in reinforcement learning. Another family of Wasserstein distances for GMMs is the Orlicz–Wasserstein distance [26].

8. Compare EM and MMOT for OCIL

8.1. Computational Complexity and Profiling Surrogates of MOOT

Let d be the latent dimensionality, K the number of centroids (mixture components) per class, and B the mini-batch size for updating a given class c. We use diagonal covariances (as in Eq. (4)), so all Gaussian operations are $\mathcal{O}(d)$ per component. One MMOT update for class c (Alg. 2) consists of: (i) sampling via the reparameterization trick and Gumbel–Softmax, (ii) evaluating and differentiating the entropic OT dual objective in Eq. (8) (or its online variant Eq. (9)), and (iii) updating the mixture parameters $\{\pi_{k,c}, \mu_{k,c}, \sigma_{k,c}^2\}_{k=1}^{K}$.

- **(i) Sampling cost (reparameterization + Gumbel–Softmax).** For each sample we form $z_k = \mu_{k,c} + \epsilon_k \sigma_{k,c}$ for all $k \in [1, \ldots, K]$ and compute relaxed mixture weights y_k via Gumbel–Softmax, then aggregate $\tilde{z} = \sum_{k=1}^{K} y_k z_k$. This requires $\mathcal{O}(BKd)$ floating-point operations and $\mathcal{O}(BK)$ memory for logits and weights.
- **(ii) Dual OT evaluation and gradients.** The dual objective in Eq. (8) is an expectation over P_c and Q_c. With a Monte-Carlo estimator using the current mini-batch as samples from P_c and one $\tilde{z}$ per data point from Q_c (the unbiased single-sample estimator in Eq. (9)), we evaluate $\phi(f_\theta(x))$, $\tilde{\phi}(\tilde{z})$, and pairwise distances $d(f_\theta(x), \tilde{z})$. Using Mahalanobis with diagonal covariance, distance evaluation is $\mathcal{O}(d)$ per pair. With one-to-one pairing, the total is $\mathcal{O}(Bd)$; with S negatives per point, it becomes $\mathcal{O}(SBd)$. Back-propagation through the Kantorovich network ϕ (a small MLP) costs $\mathcal{O}(B)$ per pass; repeating T_ϕ times per batch (Alg. 2, line 2) yields $\mathcal{O}(T_\phi B)$.
- **(iii) Mixture-parameter updates.** Gradients for $\{\pi, \mu, \sigma\}$ are linear in K and d, i.e. $\mathcal{O}(BKd)$, matching the sampling cost.

Overall complexity. A single MMOT update for one class therefore has

$$T_{\text{MMOT}} = \mathcal{O}(T_\phi B + BKd + SBd)\,,$$
$$M_{\text{MMOT}} = \mathcal{O}(Bd + Kd)\,,$$

since we never materialize a dense $B \times K$ responsibility matrix. Here T_ϕ (typically small) denotes the number of dual-network updates, and S the number of additional negatives (often $S \leq 1$).

8.2. Comparison with EM.

The classical EM algorithm (E-step + M-step) for diagonal GMMs evaluates all K component likelihoods for each of the B points and updates sufficient statistics; both steps are $\mathcal{O}(BKd)$ per iteration, repeated $I_{\text{EM}} \gg 1$ times until convergence:

$$T_{\text{EM}} = \mathcal{O}(I_{\text{EM}} BKd)\,, \qquad M_{\text{EM}} = \mathcal{O}(BK + Kd)\,,$$

where the $\mathcal{O}(BK)$ term stores per-point responsibilities across E/M steps.

Centroid-count and drift sensitivity. Both EM and MMOT scale linearly in K for diagonal Gaussians. However, MMOT avoids the inner-loop factor I_{EM} and the $B \times K$ responsibility tensor, yielding lower memory use and better stability under the continual feature drift characteristic of OCIL.

$\Rightarrow$ **Overall**, we have the complexity summary (per class, per batch) as follow

Method	Time	Memory
EM	$\mathcal{O}(I_{\text{EM}}\, BKd)$	$\mathcal{O}(BK + Kd)$
MMOT (ours)	$\mathcal{O}(T_\phi B + BKd + SBd)$	$\mathcal{O}(Bd + Kd)$

When I_{EM} exceeds a few iterations (as typically required for EM stability), MMOT becomes asymptotically cheaper in both computation and memory. Its linear scaling in K matches EM's, but the constants are smaller thanks to reparameterized sampling and diagonal Mahalanobis distances. Moreover, MMOT's single-pass stochastic updates make it better suited to streaming and non-stationary data in OCIL.

9. Implementation Details

9.1. Datasets:

As detailed in Section 5 (main text), we employ four datasets to evaluate our method's performance. These original datasets are segmented into various tasks with distinct classes. Below are the specifics regarding the dataset division and task assignments:

- **Tiny-ImageNet** consists of 200 classes, providing 100,000 training samples and 10,000 test samples, with images sized at 64 × 64 pixels. It is divided into 100 non-overlapping tasks, each containing two classes.
- **CIFAR100** includes 100 classes, offering 50,000 training samples and 10,000 test samples, also at 32 × 32 pixels. This dataset is split into 10 separate tasks, with 10 classes per task.
- **CIFAR10** contains 10 classes, with 50,000 training samples and 10,000 test samples, all sized at 32 × 32 pixels. For our experiments, it is divided into five non-overlapping tasks, each featuring two classes.
- **MNIST** consisting of 60,000 training samples and 10,000 test samples of handwritten digits (0 through 9). Each image is 28 × 28 pixels in size. For our experiment, it is splitted into 5 disjoint subsets, corresponding to 5 tasks, each of which consists of 2 classes.

For the streaming input data, we set the batch size to 10, and for the samples drawn from the buffer, the batch size is set to 64. We also employ data augmentation strategy for our method and baselines, as describe in [71].

9.2. Model Architectures:

For the experiments on MNIST dataset, we use a simple MLP neural network with 2 hidden layers of 400 units. While a slim version of ResNet-18 will be used to evaluate on three remaining datasets as commonly used in other recent state-of-the-art baselines [61, 72].

9.3. Evaluation and metrics

We used the following metrics to evaluate:

- **Average accuracy** ($\mathcal{A}_T$): Averaged test accuracy of all tasks after completing learning T task.

$$\mathcal{A}_T = \frac{1}{T}\sum_{i=1}^{T} a_i,$$

  where a_i is the accuracy at the end of the i^{th} task.
- **Average forgetting** ($\mathcal{F}_T$): The averaged gap between the highest recorded and final task accuracy at the end of continual learning on T tasks.

$$\mathcal{F}_T = \frac{1}{T-1}\sum_{i=1}^{T-1} f_i,$$

  where f_i is forgetting of task i^{th} after learning task T.

The experiments were performed over multiple runs, each with varying sequences of incoming classes. We provide the mean and standard deviation to illustrate the robustness of our results across different class orderings and random seeds.

10. Additional Experiments

10.1. Performance comparison on MNIST

Table 4 further provides a comparison between our method (OTC) and the three strongest baselines GSA, MOSE, and BiC+AC on dataset MNIST. The results show that our method consistently outperforms all these baselines in both average accuracy (up to 2.4 %) and forgetting (up to 1.6 %).

Method	M = 0.5k		M = 1.5k	
	Acc ↑	Forgetting ↓	Acc ↑	Forgetting ↓
GSA	92.7 ± 0.3	3.3 ± 0.2	96.8 ± 0.1	1.8 ± 0.2
MOSE	91.2 ± 0.4	4.0 ± 0.5	96.9 ± 0.2	1.2 ± 0.3
BiC+AC	93.0 ± 0.5	2.8 ± 0.6	97.2 ± 0.4	1.2 ± 0.3
OTC	**93.6 ± 0.3**	**2.4 ± 0.4**	**97.7 ± 0.4**	**1.1 ± 0.4**

Table 4. Evaluations on MNIST.

Method	M = 200	M = 500	M = 5120	M = 200	M = 500	M = 5120
DER++	64.88	72.70	85.24	18.66	28.70	51.20
GeoDL	49.20	61.83	85.91	13.38	23.06	54.57
Co2L	65.57	74.26	84.27	18.85	24.45	46.18
OTC (Ours)	**67.05**	**76.45**	**87.03**	**25.22**	**33.18**	**57.32**
	a) CIFAR10			b) CIFAR100		

Table 5. Average Accuracy (↑) in the offline setting of CIL, M: buffer size.

10.2. Offline setting

Table 5 examines the behavior of OTC in the offline setting of Class Incremental Learning when compared with some typical offline methods, including DER++ [5], GeoDL [60], and Co2L [6]. The results show that OTC demonstrates its superiority across all considered cases, with the most significant gap can be more than 6% compared to the strongest considered baseline. This demonstrates our effective application of our method in both online and offline setting of Class Incremental Learning problem.