Variational Data-Free Knowledge Distillation for Continual Learning (Appendix)

Xiaorong Li, Shipeng Wang, Jian Sun(✉), *Member, IEEE,* Zongben Xu

Appendix A
Proof of the Variational Lower Bound

In this Appendix, we prove the inequality (1) in Sec. 3.1.1, explaining the reason why $H(\mathbf{t}_i) + \mathbb{E}_{\mathbf{t}_i,\mathbf{s}_i}[\log q(\mathbf{t}_i|\mathbf{s}_i)]$ is the variational lower bound of mutual information $I(\mathbf{t}_i;\mathbf{s}_i)$.

$$\begin{aligned} I(\mathbf{t}_i;\mathbf{s}_i) &= H(\mathbf{t}_i) - H(\mathbf{t}_i|\mathbf{s}_i) \\ &= H(\mathbf{t}_i) + \mathbb{E}_{\mathbf{t}_i,\mathbf{s}_i}[\log p(\mathbf{t}_i|\mathbf{s}_i)] \\ &= H(\mathbf{t}_i) + \mathbb{E}_{\mathbf{t}_i,\mathbf{s}_i}[\log p(\mathbf{t}_i|\mathbf{s}_i)] \\ &\quad - \mathbb{E}_{\mathbf{t}_i,\mathbf{s}_i}[\log q(\mathbf{t}_i|\mathbf{s}_i)] + \mathbb{E}_{\mathbf{t}_i,\mathbf{s}_i}[\log q(\mathbf{t}_i|\mathbf{s}_i)] \\ &= H(\mathbf{t}_i) + \mathbb{E}_{\mathbf{t}_i,\mathbf{s}_i}[\log q(\mathbf{t}_i|\mathbf{s}_i)] \\ &\quad + \mathbb{E}_{\mathbf{s}_i}[D_{KL}(p(\mathbf{t}_i|\mathbf{s}_i)||q(\mathbf{t}_i|\mathbf{s}_i))] \\ &\geq H(\mathbf{t}_i) + \mathbb{E}_{\mathbf{t}_i,\mathbf{s}_i}[\log q(\mathbf{t}_i|\mathbf{s}_i)], \end{aligned} \tag{1}$$

where $D_{KL}(\cdot,\cdot)$ is Kullback-Leiber divergence. The last inequality holds due to the non-negativity of $D_{KL}(\cdot,\cdot)$.

Appendix B
Details on Compressing Gradients of BN and fully connected layers

For the batch normalization (BN) layer with n feature channels, each channel is equipped with an affine transformation with two parameters, i.e., scaling weight and bias. Therefore, the gradients of scaling weight and bias in BN layer are both with the size of $n \times C$, where C is the number of classes. We respectively compress the gradients of scaling weight and bias by SVD without using average operation.

For the fully connected layer, we compress its gradients by SVD without average operation. Specifically, the gradients of the fully connected layer are of size $n_{out} \times n_{in} \times C$, where n_{out} (n_{in}) is the output (input) dimension of a fully connected layer and C is the number of classes. The gradients can be regarded as n_{out} matrices with size $n_{in} \times C$. We apply SVD on each matrix of size $n_{in} \times C$ and obtain corresponding SVD decomposition (i.e., singular values with size r, left singular vectors with size $n_{in} \times r$ and right singular vectors with size $C \times r$, $r \ll C$).

Appendix C
Implementation Details

In this Appendix, we provide the setting of hyperparameters of VDFD in all experiments. For all experiments, we only save the largest singular value and corresponding singular vectors in the compression phase, i.e., $r = 1$. Besides, we adopt a two layers GCN, i.e., $M = 2$.

C.1 Continual Learning for Image Classification

Task-incremental setting. We use Adam optimizer in task-incremental setting. The settings of hyperparameters are shown in Table 1, where "10-CIFAR", "20-CIFAR", "25-Tiny", "10-Sub" and "Five" denote 10-split CIFAR-100, 20-split CIFAR-100, 25-split TinyImageNet, 10-split SubImageNet and 5-Datasets respectively. The network is trained 30 epochs in total for 5-Datasets and 80 epochs for others. The initial learning rate reduces with a multiplier of 0.5 at epoch 15 and 20 for 5-Datasets and reduces with the same multiplier at epoch 30 and 60 for others.

TABLE 1
Hyperparameters of our VDFD in task-incremental setting. "10-CIFAR", "20-CIFAR", "25-Tiny", "10-Sub" and "Five" denote 10-split CIFAR-100, 20-split CIFAR-100, 25-split TinyImageNet, 10-split SubImageNet and 5-Datasets respectively.

Dataset	Learning rate	Learning rate of GCN	Weight decay	Batch size	λ
10-CIFAR	1×10^{-4}	5×10^{-4}	5×10^{-4}	32	5
20-CIFAR	1×10^{-4}	7×10^{-4}	5×10^{-4}	32	5
25-Tiny	5×10^{-5}	5×10^{-4}	1×10^{-5}	16	200
10-Sub	1×10^{-4}	1×10^{-4}	1×10^{-5}	16	150
Five	5×10^{-4}	5×10^{-4}	5×10^{-5}	16	80

Class-incremental setting. For all experiments in class-incremental setting, we use Adam optimizer and train the network for 100 epochs. The learning rates of the network and GCN start from 1×10^{-4} and decay 0.5 at epoch 50. The batch size and weight decay are set to 16 and 7×10^{-4}. We set the hyperparameter balancing the plasticity and stability across all seen tasks to $\lambda = 50$ for CIFAR-100 and $\lambda = 100$ for TinyImageNet respectively.

C.2 Continual Learning for Semantic Segmentation

We use SGD optimizer with momentum of 0.9 for all competitors in all experiments. The network is trained for 50 epochs and the batch size is set to 24. We train the network with an initial learning rate of 0.02 for the first task and 0.001 for the subsequent tasks. We set the initial learning rate of GCN to 0.001. The learning rates of the network and GCN are adjusted by the polynomial learning rate decay scheduler. We set the hyperparameter balancing the plasticity and stability across all seen tasks to $\lambda = 10$.

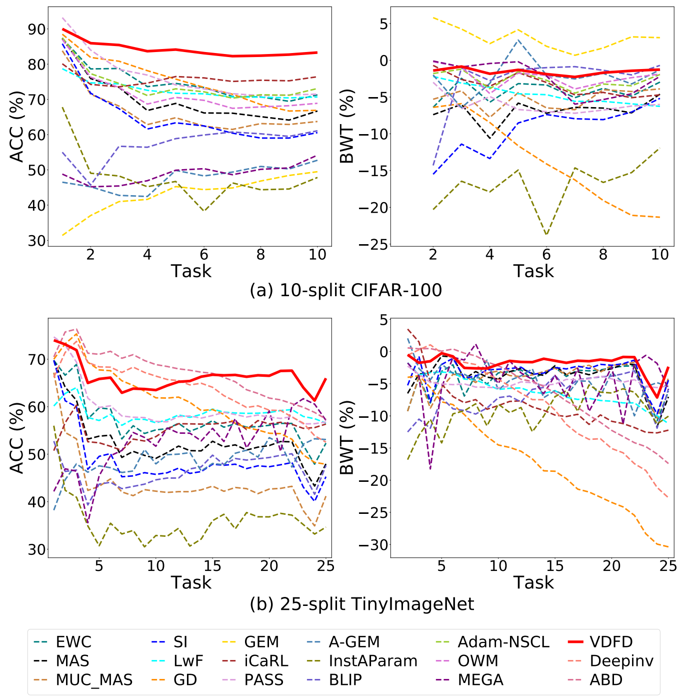

Fig. 1. The curves of ACC and BWT values w.r.t. the number of tasks on 10-split CIFAR-100 and 25-split TinyImageNet by different methods.

Appendix D
Comparison on ACC and BWT Values

In this Appendix, we show ACC and BWT values of compared methods in the whole task sequence.

10-split CIFAR-100. Comparative results of 10-split CIFAR-100 are visualized in Fig. 1(a). As shown in the left of Fig. 1(a), VDFD achieves the best ACC consistently with the coming of new tasks. Though achieving better BWT values, GEM achieves worse ACC values than ours in the whole sequence as illustrated in the left of Fig. 1(a). Additionally, Adam-NSCL, OWM and BLIP show comparable BWT values according to the right of Fig. 1(a), but they fail to achieve similar ACC as ours, which is illustrated in the left of Fig. 1(a).

25-split TinyImageNet. Comparative results of 25-split TinyImageNet are shown in Fig. 1(b). As suggested by Fig. 1(b), VDFD achieves better performance than compared methods considering both ACC and BWT values. Though Deepinv, GD and ABD perform marginally better on ACC in the beginning, their ACC and BWT values degrade dramatically when the number of tasks grows larger. Therefore, they forget more, which implicitly indicates that synthesized data or coreset of previous tasks may hurt the network stability.

Appendix E
Effect of Different Sampling Strategies on Distillation-based Methods

In this Appendix, we discuss the effect of different sampling strategies on two compared distillation-based methods, GD [1] and iCaRL [2], which require storing a coreset containing training data of previous tasks. Overall, GD with herding sampling and iCaRL with herding sampling achieve the best ACC and BWT values among all sampling strategies. However, they still fail to achieve as good performance as our VDFD.

GD employs a newly proposed distillation with the saved coreset as input. iCaRL uses the coreset to compute the prototypes of old classes which are further utilized for classification. The saved coreset is also used for distillation in iCaRL. We utilize the sampling strategies proposed in [3] and [4] to select training samples of previous tasks to be saved, including random sampling, herding sampling, reservoir sampling, entropy-based sampling and plane distance-based sampling strategies.

Specifically, in the random sampling strategy, training data of previous tasks are selected randomly. In the herding sampling strategy, the samples are selected when their feature representations are closer to one of the prototypes of all seen classes. The prototype of a class is computed by averaging all the feature vectors of the corresponding class. In the reservoir sampling strategy, each training data is sampled from an unknown size data stream with a certain probability in a single pass manner, where the probability equals $\frac{k}{n}$ with k as the size of coreset and n as the number of observed data. In the entropy-based sampling strategy, the samples with higher entropy of the output softmax distribution are selected, which measures the uncertainty of a sample. In the plane distance-based sampling strategy, the samples closer to the decision boundary are selected. The distance $d(x_i)$ between a sample x_i with label y_i and the decision boundary is measured by $d(x_i) = \phi(x_i)^\top w^{y_i}$, where $\phi(\cdot)$ is the learned feature extractor and w^{y_i} is the parameter of the last fully connected layer of class y_i.

TABLE 2
The comparisons of ACC and BWT values by GD and iCaRL with different sampling strategies on 20 split CIFAR-100.

Sampling strategy	Metrics	GD	iCaRL
Random	ACC (%)	78.16	74.40
	BWT (%)	-14.39	-6.83
Herding	ACC (%)	79.06	75.75
	BWT (%)	-13.20	-6.08
Reservoir	ACC (%)	78.97	74.78
	BWT (%)	-13.41	-7.22
Entropy	ACC (%)	75.97	58.04
	BWT (%)	-16.31	-5.42
Plane distance	ACC (%)	76.16	49.82
	BWT (%)	-16.05	-2.81

The experimental results of iCaRL and GD using different sampling strategies on 20-split CIFAR-100 are shown in Table 2. According to Table 2, GD and iCaRL with different sampling strategies show diverse performance, suggesting that a proper sampling strategy is helpful to improve performance. Among GD with different sampling strategies, herding sampling obtains the best ACC and BWT values. GD with reservoir sampling achieves the second best ACC and BWT values, which is marginally higher than GD with random sampling. GD with entropy-based sampling and GD with plane distance-based sampling perform worse than GD with other sampling strategies in this experiment, indicating that selecting the boundary samples may lessen the ability of distillation. As for iCaRL, using herding sampling is also the best choice, since it achieves the highest ACC value and comparable BWT value among all sampling strategies. iCaRL with reservoir sampling and iCaRL with random sampling obtain nearly similar ACC and BWT values. However, iCaRL with entropy-based sampling and iCaRL with plane distance-based sampling perform much worse than iCaRL with other sampling strategies. It suggests that using boundary samples as prototypes for classification may lower the performance of the network. It is worth mentioning that our VDFD (w/o SSL) achieves ACC value of 80.97% and BWT value of -4.79%, and our VDFD achieves ACC value of 85.84% and BWT value of -1.53% in this experiment.

APPENDIX F COMPARISON WITH ARCHITECTURE-BASED METHODS

In this Appendix, we compare our method with several architecture-based methods including RPSNet [5], DER [6] and DyTox [7]. Since we focus on the setting where the training data of previous tasks are unavailable, we further consider their variants which do not replay previous data. The variants that do not replay previous data are denoted as w/o replay.

Comparison with RPSNet. To mitigate catastrophic forgetting, RPSNet progressively expands the network and leverages replayed data as well as knowledge distillation. Comparative results on CIFAR-100 are shown in Fig. 2, where the results of RPSNet are from its original paper. According to Fig. 2, our VDFD and VDFD (w/o SSL) perform better than RPSNet w/o replay but worse than RPSNet. However, RPSNet dynamically expands the network with the coming of new tasks, which has 72.26×10^6 parameters at the end of the task sequence. While our method adopts ResNet-18 as the backbone with 11.2×10^6 parameters. Besides, it can be seen that the performance of RPSNet heavily relies on the replayed data, which restricts it from directly applying to the setting where the training data of previous tasks are inaccessible.

Comparison with DER. When learning a new training task, DER adds a new learnable feature extractor for its network architecture and freezes the previously learned feature extractors, which attempt to integrate new knowledge and retain learned knowledge. Besides, it introduces channel-level masks to prune the network and replays previous data. Since the pruning strategy utilized in DER is not published in their official implementation[1], we further compare its variant without pruning and replay, which is denoted as DER w/o P&replay. Comparative results on CIFAR-100 are shown in Fig. 2, where the results of DER and DER w/o P are from its original paper. As shown in Fig. 2, our VDFD and VDFD (w/o SSL) perform better than DER w/o P&replay, which shows the superiority of our method in the setting where the training data of previous tasks are inaccessible. Though DER and DER w/o P perform better than VDFD and VDFD (w/o SSL), they adopt a dynamically expanding network with ResNet-18 as the basic network. While our method adopts ResNet-18 as the backbone with 11.2×10^6 parameters. After learning 10 tasks on CIFAR-100, the network utilized in DER w/o P has 112.27×10^6 parameters. The number of network parameters of DER at the end of task sequence is not available, since DER reports the average number of parameters over all tasks, which is lower than the number of network parameters at the end of task sequence.

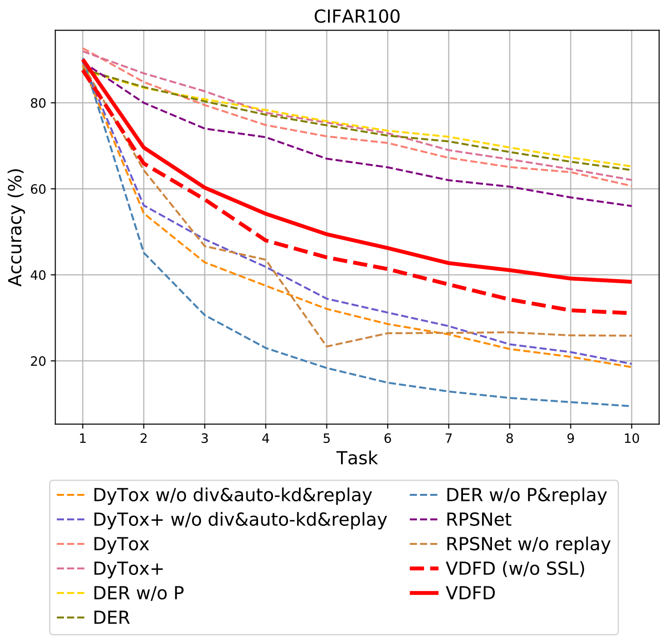

Fig. 2. The comparisons of accuracies after learning each task on CIFAR-100. All of the methods that outperform our VDFD and VDFD (w/o SSL) rely on replaying the training data from previous tasks. When these methods do not replay previous data, their performance is lower than ours.

Comparison with DyTox. The authors of DyTox propose a transformer architecture with dynamic expansion of task tokens. Meanwhile, DyTox adopts knowledge distillation, rehearsal and an auxiliary classifier to mitigate catastrophic forgetting. Besides, its improved version DyTox+ additionally adopts MixUp [10] strategy which utilizes new samples obtained by linearly interpolating existing samples. We further compare our method with various variants of DyTox, since we empirically find that some techniques utilized in DyTox produce a negative effect on performance when it does not replay previous data. We provide a brief introduction of the variants in the following.

- w/o div: it means that removing the divergence loss

1. https://github.com/Rhyssiyan/DER-ClassIL.pytorch

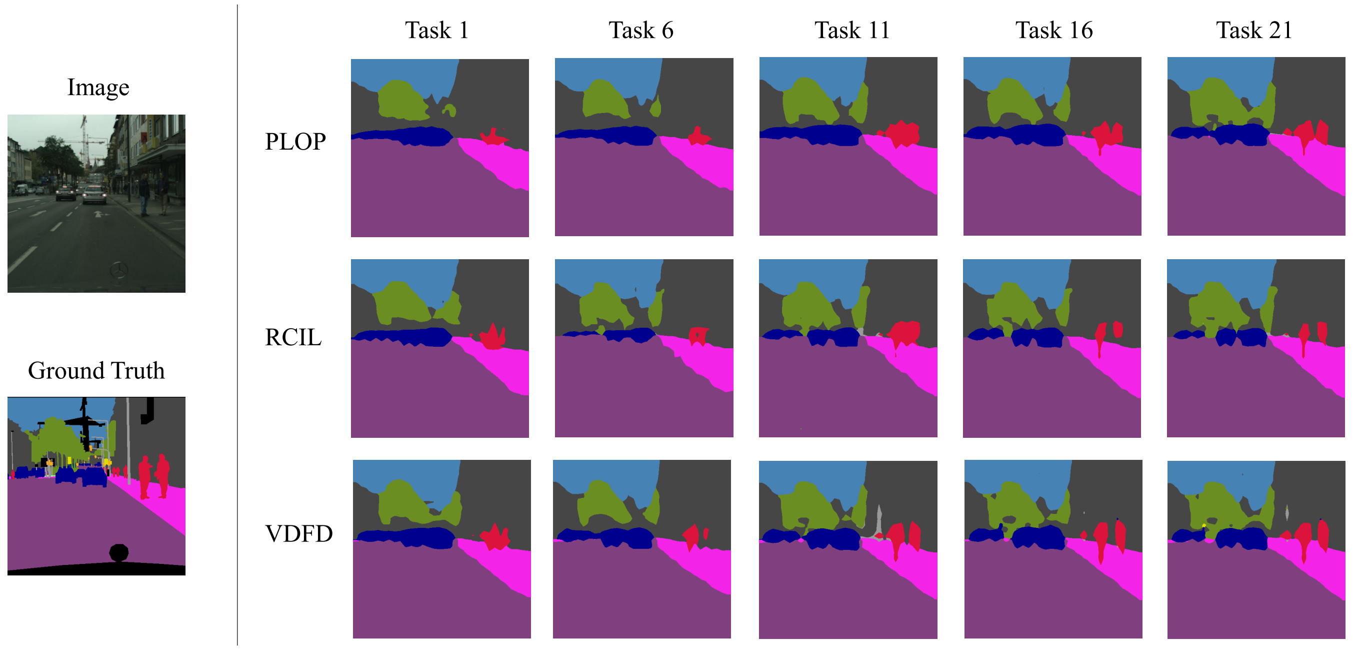

Fig. 3. Visualization of predictions of PLOP [8], RCIL [9] and VDFD after learning tasks $\mathcal{T}_1$, $\mathcal{T}_6$, $\mathcal{T}_{11}$, $\mathcal{T}_{16}$ and $\mathcal{T}_{21}$ in the setting "1-1" of Cityscapes.

$\mathcal{L}_{div}$ in Eq. (9) of the paper of DyTox[2]. The divergence loss is computed on the auxiliary classifier which predicts the probabilities of the current task classes and an extra class representing all previous classes.

- w/o auto-kd: we set the coefficients of classification loss $\mathcal{L}_{clf}$ and distillation loss $\mathcal{L}_{kd}$ as 1 and 50, rather than automatically determining the coefficient (i.e., α in Eq. (9) of the paper of DyTox) according to the number of seen classes.

Comparative results on CIFAR-100 are shown in Fig. 4, where the results of DyTox, DyTox+ are from its original paper. According to Fig. 4, our VDFD and VDFD w/o SSL perform better than all variants of DyTox without replay but worse than DyTox, DyTox+. It indicates that when the training data of previous tasks are inaccessible, our method is preferred. Furthermore, the performance of DyTox and DyTox+ is heavily dependent on the replayed data.

Among the variants of DyTox without replay, DyTox w/o div&replay (DyTox+ w/o div&replay) performs better than DyTox w/o replay (DyTox+ w/o replay). It shows that the divergence loss fails to improve performance when the training data of previous tasks are not available. DyTox w/o div&auto-kd&replay (DyTox+ w/o div&auto-kd&replay) performs better than DyTox w/o div&replay (DyTox+ w/o div&replay). It indicates that increasing the strength of distillation loss is reasonable when replaying previous data is not allowed. We report the results of DyTox w/o div&auto-kd&replay and DyTox+ w/o div&auto-kd&replay in Fig. 5 of the manuscript, they obtain the highest results.

2. Eq. (9): $\mathcal{L} = (1-\alpha)\mathcal{L}_{clf} + \alpha\mathcal{L}_{kd} + \lambda\mathcal{L}_{div}$, where $\mathcal{L}_{clf}$ and $\mathcal{L}_{kd}$ are classification loss and distillation loss, α and λ are hyperparameters.

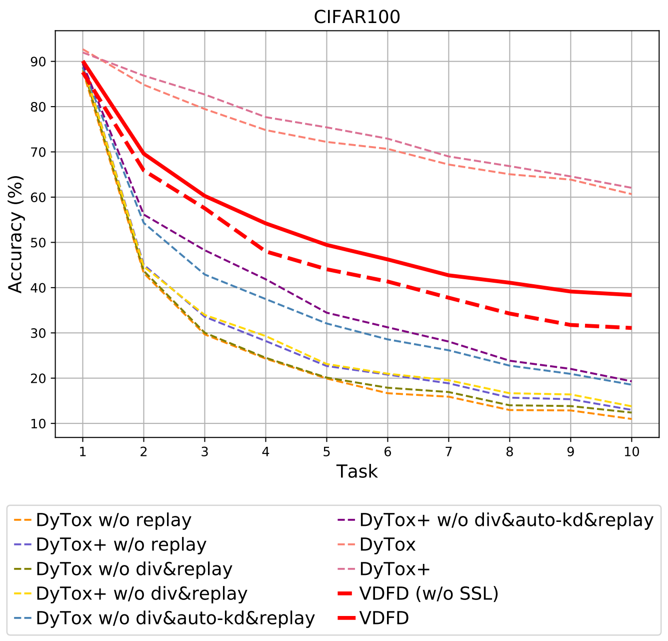

Fig. 4. The comparisons of accuracies after learning each task on CIFAR-100. All of the methods that outperform our VDFD and VDFD (w/o SSL) rely on replaying the training data from previous tasks. When these methods do not replay previous data, their performance is lower than ours.

Appendix G Visualization on Continual Semantic Segmentation

We visualize the predictions of PLOP [8], RCIL [9] and our VDFD on the setting "1-1" of Cityscapes in Fig. 3, where the predictions are obtained after learning tasks $\mathcal{T}_1$, $\mathcal{T}_6$, $\mathcal{T}_{11}$, $\mathcal{T}_{16}$ and $\mathcal{T}_{21}$. According to Fig. 3, the segmentation map obtained by our VDFD does not dramatically change with the increase of the number of learned tasks, which indicates that our VDFD well retains the stability of the network. Compared to PLOP and RCIL, our VDFD generates the segmentation with more details at the last task $\mathcal{T}_{21}$ (e.g., person and pole).

Appendix H VDFD with Differential Privacy Technique

In this Appendix, we further introduce the technique of differential privacy to our method for lessening the risk of privacy leakage of gradients. Overall, VDFD with differential privacy technique achieves marginally lower performance than VDFD when applying a small noise level.

Differential privacy [11], [12] is a standard definition for privacy guarantee. Intuitively, it guarantees that a randomized algorithm behaves similarly on adjacent input datasets. In our experiments, each training dataset consists of image-label pairs. We say that two datasets are adjacent if one image-label pair is present in one dataset and absent in the other dataset. The definition of differential privacy is described in the following.

Definition 1 (Differential Privacy [12]). *A randomized mechanism $\mathcal{M}$ with domain $\mathcal{D}$ is (ϵ, δ)-differential privacy if for all $\mathcal{S} \subseteq Range(\mathcal{M})$ and for any two adjacent inputs $d, d' \in \mathcal{D}$:*

$$Pr[\mathcal{M}(d) \in \mathcal{S}] \leq \exp(\epsilon) Pr[\mathcal{M}(d') \in \mathcal{S}] + \delta,$$

where $Pr[\cdot]$ denotes the probability.

A common paradigm to ensure a deterministic function q with differential privacy guarantee is adding noise to the output of q. For instance, Gaussian mechanism [12] in the following Definition 2.

Definition 2 (Gaussian mechanism [12]). *Given a deterministic function q with domain $\mathcal{D}$, Gaussian mechanism is defined as:*

$$\mathcal{M}(d) = q(d) + n,$$

where n is a random variable drawn from Gaussian distribution $\mathcal{N}(0, \sigma^2)$.

It can be proved that Gaussian mechanism is (ϵ, δ)-differential privacy [12, Theorem 3.22]. To prevent leakages of gradient information in our method, we employ Gaussian mechanism on the query function q that inputs the training data and returns the corresponding gradients of network parameters. Once we obtain the output gradients using Gaussian mechanism, we apply the proposed compression method described in Sec. 3.1.3 of the paper on the gradients. In our experiments, the noise of Gaussian mechanism is drawn from $\mathcal{N}(0, s\sigma^2)$, which is simultaneously calibrated by the noise level s and the variance of gradients σ^2. The variance of gradients $\sigma^2 \in \mathbb{R}^{|w| \times C}$ is estimated on the whole training dataset, where $|w|$ and C denote the dimension of network parameters and the number of classes respectively. This ensures that the noise is adaptively calibrated by the scale of the gradients.

The comparisons of ACC and BWT values on 10-split CIFAR-100 and 25-split TinyImageNet with different noise levels are shown in Table 3. As shown in Table 3, the ACC values decrease with the increase of noise levels. The BWT values with adding noise are lower than the BWT value without adding noise.

TABLE 3
The comparisons of ACC and BWT values on 10-split CIFAR-100 and 25-split TinyImageNet with different noise levels s. Values in bold and underlined are respectively the best and the second best results.

Noise level	10-split CIFAR-100		25-split TinyImageNet	
	ACC (%)	BWT(%)	ACC(%)	BWT(%)
$s = 0$	**79.23**	**-2.93**	**60.58**	**-4.60**
$s = 0.001$	<u>79.12</u>	-3.55	<u>60.49</u>	<u>-4.80</u>
$s = 0.01$	78.88	<u>-3.52</u>	60.24	-5.01
$s = 0.1$	78.68	-3.94	60.08	-5.06
$s = 0.2$	78.27	-3.50	59.20	-5.93

References

[1] K. Lee, K. Lee, J. Shin, and H. Lee, "Overcoming catastrophic forgetting with unlabeled data in the wild," in *Int. Conf. Comput. Vis.*, 2019, pp. 312–321.

[2] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017, pp. 2001–2010.

[3] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato, "On tiny episodic memories in continual learning," *arXiv preprint arXiv:1902.10486*, 2019.

[4] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *Eur. Conf. Comput. Vis.*, 2018, pp. 532–547.

[5] J. Rajasegaran, M. Hayat, S. H. Khan, F. S. Khan, and L. Shao, "Random path selection for continual learning," in *Adv. Neural Inform. Process. Syst.*, 2019, pp. 12 669–12 679.

[6] S. Yan, J. Xie, and X. He, "Der: Dynamically expandable representation for class incremental learning," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021, pp. 3014–3023.

[7] A. Douillard, A. Ramé, G. Couairon, and M. Cord, "Dytox: Transformers for continual learning with dynamic token expansion," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022, pp. 9285–9295.

[8] A. Douillard, Y. Chen, A. Dapogny, and M. Cord, "Plop: Learning without forgetting for continual semantic segmentation," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021, pp. 4040–4050.

[9] C.-B. Zhang, J.-W. Xiao, X. Liu, Y.-C. Chen, and M.-M. Cheng, "Representation compensation networks for continual semantic segmentation," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022, pp. 7053–7064.

[10] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *Int. Conf. Learn. Represent.*, 2018.

[11] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *ACM Conf. Comput. Commun. Security*, 2016, pp. 308–318.

[12] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.