TRIPLET LEARNING OF TASK REPRESENTATIONS IN LATENT SPACE FOR CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual learning is a mechanism where a model is trained on tasks sequentially and learns current task while retaining the knowledge of previous tasks. Existing methods that utilize latent space include rehearsal with latent spaces and latent space partitioning. However, latent space overlapping can cause interference between knowledge of different tasks, leading to performance drop on specific metrics, e.g., classification accuracy or quality of image reconstruction. To solve this problem, we propose a method of training autoencoder with triplet loss applied to partition its latent space. We denote the output of the encoder and some manually chosen layers of the decoder as original latent space and intermediate latent space, respectively. Specifically, we use triplet loss in the intermediate latent space to mitigate the overlapping by clustering latent variables of same classes and separating that of different classes. We test our method on several datasets, including MNIST, FashionMNIST, Omniglot, and CelebA. From those experiments, our model shows an FID of 19 on MNIST and a recall of 0.272 on CelebA, which are better than the state-of-the-art models when trained in similar setups. Qualitatively, we confirm better partitioning results of the proposed method by comparing the visualization of latent space with other latent space methods.

1 INTRODUCTION

Considering the real learning scenario that learns various data sequentially, many researchers have studied on continual learning that learns data incrementally. In continual learning setup, we cannot train a model on all training data repeatedly. Instead, the model is trained with the tasks coming one by one and in some cases the previous task is not available after the model learns it due to the reasons such as limitation of memory, privacy issue, etc. Under this setup, the model tends to forget previously learned knowledge about previous tasks after learning new tasks, which is called catastrophic forgetting. Thus, continual learning aims to make the model able to learn the new task without suffering from the forgetting problem. Catastrophic forgetting can be widely observed in all kinds of deep neural networks when trained under continual learning setup.

To overcome the catastrophic forgetting, various methods that use generative model for replaying previous tasks are investigated, and show high performance in continual learning setup. In these methods, the generative model needs to learn a new distribution of new task data, without disturbing the distribution of old task data. Ideally, a model can learn the new task better based on the learned knowledge, but the possible overlapping between distributions of knowledge can result in performance drop.

To preserve the learned knowledge, we focus on latent space, which consists of variables from the output of the layers. The knowledge that a human learned are somehow stored in the direct connections and synapses between neurons in the brain. Similarly, the knowledge learned by a neural network can be reflected through latent space. When we visualize the latent space of a well-trained classification model on dataset like MNIST, we can find the latent space of different classes of data are well clustered. Otherwise, overlapping of latent spaces or dispersed latent variables can be found, which means the knowledge between different data interfere with each other, causing the forgetting problem. Overlapping is not a problem only of image generation. Essentially, any method that embeds data to an embedding space can suffer from this problem.

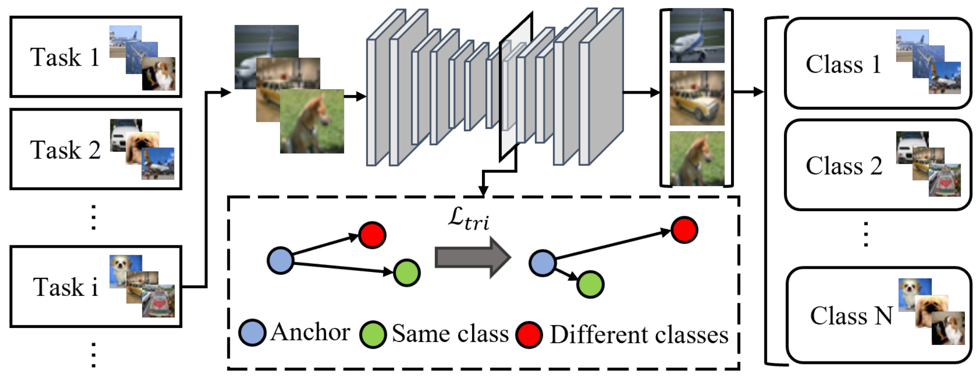


Figure 1: Concept explained: apply triplet loss to image generation under continual learning setup.

Inspired by the idea of triplet loss ((Schroff et al., 2015)), which is the most widely used loss function in deep metric learning, we propose a method for partitioning latent space of the generative model under continual learning setup. Let x_1, x_2 be the data samples that have the same label y_1, and x_3 has a different label y_3, and a model M which converts data x to latent variable z. The goal of partitioning latent spaces is training a model M which satisfies that the value of $||z_1, z_2||$ (i.e. distance) is lower than $||z_1, z_3||$ or $||z_2, z_3||$. The loss function can be defined as follows:

$$\mathcal{L}_{\text{tri}} = max\left(0, ||z_1, z_2|| - ||z_1, z_3|| + margin\right), \tag{1}$$

where $margin$ is a hyperparameter that stimulates the model to adjust the distance between between data pair of different classes longer than the distance between data pair of same class for at least a distance of $margin$.

In our study(Figure 1), we use an autoencoder as a model; we use two latent spaces: the output of the encoder (called as the original latent space) and the output of a manually chosen layer in the decoder (called as the common latent space). In the process of continual learning, when the data of the new task reaches the middle layer, the parameters of the middle layer should be updated to ensure that the newly learned knowledge cannot be confused with the previous knowledge, or cause confusion between the knowledge learned on the previous tasks. To solve the potential confusion problem, we use triplet loss to adjust the latent variables in the common latent space to make the latent variables of the same class of data close to each other, forming a non-scattered latent space. At the same time, we can increase the distance between latent variables of different classes of data, so that the latent spaces of different classes of data will not interfere with each other. Every time there is a new task, we perform the above operations to achieve the objective of learning the new task without losing the performance on previous tasks.

2 Related work

Regularization methods focus on how parameters are updated. In some works, authors prevent important parameters from updating significantly by their importance. The parameters that are responsible for some previous tasks are considered as important for those tasks. Therefore, preserving these parameters are necessary to mitigate performance drop on previous tasks. There are different ways to measure the importance of parameters. Zenke et al. (2017) compute the importance of certain parameter to a task by adding a little perturbation to the parameter and check how much the value of the loss function for this task changes. The more change of the loss function value, the more important the parameter is to the task, i.e. we should prevent it from changing to preserve the performance on this task. Every time there is a new task, the importance of parameters will be computed and updated. Slightly differently, Aljundi et al. (2018) consider a model as a function and computer the importance of parameters by checking the change of the model after adding a perturbation to the parameters. Aljundi et al. (2018) claim that using loss function value as the measurement can be difficult when the loss gets into a local minimum. One limitation of methods like (Aljundi et al.,

2018) and (Zenke et al., 2017) is that if all parameters are considered important to some certain task(s), no parameters will be left for learning new tasks, because update of any parameters will cause forgetting on previous tasks.

The first kind of replay method is rehearsal with real data. Just like human review what he or she learned to prepare for an exam, model needs to review the data of previous tasks when learning new tasks. In (Rebuffi et al., 2017), the model samples from each class of data of the current task and saves the samples to memory. The sampled data are chosen to best represent the mean value of its class, and will be used to When testing, the mean value of each class will be computed based on the sampled data to do the nearest-mean classification. However, re-calculation of the mean value of class in an ever-changing data stream, which is the more common scenario, can be computationally expensive. De Lange & Tuytelaars (2021) proposed a method to compute and update the prototype dynamically at a lower cost, as well as a mechanism that can keep the rate of samples of different classes remain at a specific value. However, real data replay methods may suffer from the overfitting problem because the buffered images cannot represent the whole distribution.

Another way to do replay is called pseudo replay, where a generative model is trained together with the main model. The generative model is used to generate pseudo data of previous tasks, which will be used to train the main model with current task data. Deep Generative Replay (Shin et al., 2017) uses a generator to generate pseudo replay data for the classification model and achieves close performance compared to real data replay. Lavda et al. (2018) connect a VAE with a classifier, and separate the model with a pair of ever-updating teacher and student: the teacher generates data corresponding to previous tasks, the student learns both on generated data and current task data, after which the student will become a teacher for next task. However, since a generative model itself can also suffer from catastrophic forgetting, pseudo replay methods have been criticized for simply shifting the forgetting problem to a generative model instead of fundamentally solving the problem. Atkinson et al. (2018) address this issue by applying pseudo data replay on both the classifier and DCGAN ((Radford et al., 2015)), the generative model they used. Since the training of generative model itself is not an easy job, the performance of pseudo data replay methods can be limited to the performance of the generative model.

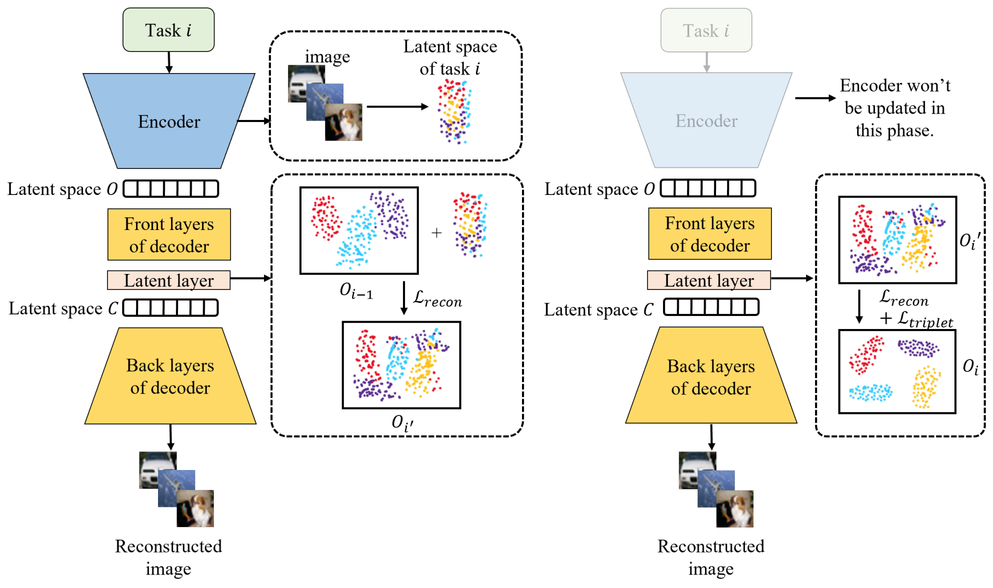
The two kinds of methods mentioned above either use real data or generated data, which can be spatially ineffective. Instead, some methods utilize the output of a certain intermediate layer of the network, i.e. latent layer, to do replay, which can save much more space than replay with real data. Latent Replay for Real-Time Continual Learning (Pellegrini et al., 2020) aims to solve the forgetting problem of classification on edge devices that only have light computational ability. Pellegrini et al. (2020) choose a layer as the latent layer, and save the output of the latent layer as patterns of tasks learned so far to an external memory. When training, layers between latent layer and output layer are set with a larger learning rate and are trained on both current data and saved patterns, while the layers between input layer and latent layer are set with a lower learning rate and are trained with only current data. But the issue is that if the model is suffering from forgetting problem, the output of some intermediate layer may contain chaotic information about the real data, i.e. not qualified to represent the real data. Thus, the replay won't be effective. The common drawback is that although these methods utilize latent spaces, there is not any straightforward optimization about the latent spaces. So, it is reasonable to think that we need a method to do operation, e.g. adjusting the distance between latent spaces or clustering latent variables, on the latent space-level, which can be achieved by using techniques of deep metric learning.

Deep metric learning has been applied to many different jobs to enhance the model's ability of representation learning. Chen et al. (2020) use contrastive learning ((Chopra et al., 2005)) to let the model learn a better representation of the data. Under unsupervised learning setup, we don't have class labels to define whether two data samples are similar or not. Therefore, Chen et al. (2020) use data augmentation: two augmentations are considered to be similar if they come from the same image, or are not similar if they come from different images. Experimental results show that (Chen et al., 2020) can help their model learn better representation, achieving close results with models trained under supervised learning setup. Jeong & Shin (2021) use the contrastive learning loss from (Chen et al., 2020) together with other loss functions to alleviate the overfitting problem of the discriminator of GAN, making it possible to apply a wider range of data augmentation. In (Jeong & Shin, 2021), besides the contrastive loss used in (Chen et al., 2020), the authors define that the data samples from real and generated data are not similar. Together with traditional classification loss, the discriminator learns better representation, and therefore can distinguish real and generated samples more effectively.

One of the methods that utilizes deep metric learning in continual learning setup is (Cha et al., 2021), whose authors applied contrastive learning to classification job under continual learning setup. Cha et al. (2021) found that the model can learn more robust representations against catastrophic forgetting when trained with contrastive loss than when trained with cross entropy. When training on tasks, samples of current task are used as anchor samples, and samples of past tasks that are saved in a memory buffer are used as negative samples. In this way, the model will not get confused about current and past tasks, which can also be considered as a way to prevent overlapping between representations of new and old tasks. What's more, to enable the model to learn transferable knowledge between tasks, Cha et al. (2021) use instance-wise relation distillation (IRD) to quantify the discrepancy between the instance-wise similarities of the representations of current and past tasks.

3 Triplet learning for latent space partitioning

In this section, we present the proposed method. To achieve better performance under continual learning setup, we focus on the partitioning of latent spaces. We use a Variational Autoencoder (VAE) as the base model, which consists of an encoder and a decoder. Inspired by (Pellegrini et al., 2020), we manually choose an intermediate layer of the decoder as the latent layer. The output of the encoder is denoted as original latent space O, and the output of the latent layer is denoted as the common latent space C, where we apply triplet learning to do latent space partitioning. Our training consists of two phases: (1) learning phase, where the model learns the current task and (2) partitioning phase, where we do latent space partitioning to let the model refine the knowledge of current and previous tasks. Through these two phases, the model learns a new task and consolidates the new knowledge together with previous ones. Figure 2 and Figure 3 showed the overview and the schematic diagram of our method.



(a) Learning phase (b) Partitioning phase

Figure 2: Overview of our method. (a) is the learning phase, where the model learns a new task through standard VAE training, with possibility of overlapping of latent spaces. (b) is the partitioning phase, where we apply triplet loss to do latent space partitioning.

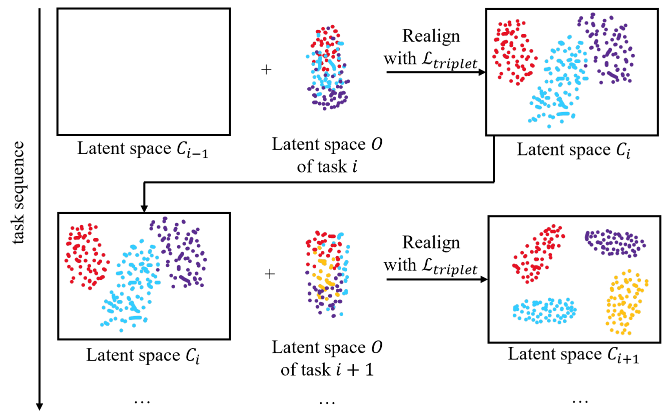

task sequence
Latent space C_{i-1}
+
Latent space O of task i
Realign with $\mathcal{L}_{triplet}$
Latent space C_i
Latent space C_i
+
Latent space O of task $i+1$
Realign with $\mathcal{L}_{triplet}$
Latent space C_{i+1}
...

Figure 3: A schematic diagram of latent space partitioning.

3.1 Formal Definitions

Basic terminologies. $X = \{x_1, x_2, \ldots x_n\}$ is the training dataset containing n images; q is the encoder that maps an image x to a latent variable λ; p is the decoder that regenerates an image x' using λ.

Latent space partitioning. Generally, the corresponding output of a data sample is denoted as latent variable, and all the latent variables of the data samples of the same class form the latent space of that class of data. What's more, we see the process that the latent layer is updated on tasks sequentially as a process of putting the latent spaces of the current task to the common latent space C. So, by latent space partitioning in common latent space C, we mean the partitioning of latent spaces of data samples of different classes.

Model version. We denote the model after partitioning phase of task i as M_i. Similarly, the original latent space O and common latent space C after learning of task i is O_i and C_i, respectively.

Distribution of two latent spaces. We denote the distribution of original latent space and common latent space as P_O and P_C.

3.2 Learning phase

In learning phase, we let the model learn the current task through regular VAE training, where we use reconstruction loss to optimize the ability to regenerate an image. When the first task comes, the model learns the task from an initialization of Gaussian distribution. After learning the first task, the model learns task i from model $M_{i-1}, i > 1$. We use the optimization proposed by Kingma & Welling (2013) to maximizes the variational lower bound of log likelihood.

$$\max_{\theta,\phi} \mathbb{E}_{q(\lambda|\mathbf{x})}[\log p(\mathbf{x} \mid \lambda)] - \eta D_{KL}(q(\lambda \mid \mathbf{x}) \| \mathcal{N}(\overrightarrow{0}, I)), \tag{2}$$

where θ,ϕ are the parameters of the encoder and decoder, respectively; and $\eta > 0$ is a hyperparameter that adjust the strength of forcing the distribution of latent variable to be a normal distribution. In our experiment, we set $\eta = 1$ in our experiment.

The learning phase is like the process of a human learns some new knowledge for the first time. Although he or she can really learn something at the first time, but the learning result may be not satisfactory enough. Therefore, we need to reinforce the newly learned knowledge while keeping memory of previously learned knowledge. Here comes the partitioning phase.

3.3 Partitioning phase

After the learning phase, the model has somehow learned the current task. However, in the meantime, overlapping of latent spaces may have happened, causing forgetting towards previous tasks.

In the partitioning phase, to solve the potential overlapping problem, we only focus on the decoder and freeze the encoder. We divide the decoder into two parts: layers between the encoder and the latent layer and layers between the latent layer and the output layer, which are called front layers and back layers, respectively. Similar to the usual training mode in transfer learning, the learning rate of the front layers is set to be one-tenth of the learning rate of the back layers, which is the warm-up stage of partitioning phase, because we don't want the lower-level latent variables of front layers to be changed too much. After the warm-up stage, the whole decoder will be trained with the same learning rate.

Besides reconstruction loss such as Binary CrossEntropy (BCE) and Mean Squared Loss (MSE), we apply triplet loss to do partitioning in the common latent space O. Triplet loss is one of the ranking losses used in deep metric learning, whose core idea is adjusting the distance of samples in the embedding space. To apply triplet loss, we need to designate the positive latent variable and latent variable for some certain anchor latent variable. Simply, we define that the latent variables of the data of the same class are the positive ones; otherwise, the negative ones. The triplet loss is defined as follows:

$$l_{tri} = \max\left(\|c_a - c_p\| - \|c_a - c_n\| + \text{margin}, 0\right), \tag{3}$$

where c_a, c_p, c_n are the anchor, positive and negative latent variables (i.e. latent spaces of data of different classes) in the common latent space, and margin is used to control the strength of partitioning.

As for getting the latent spaces, we just use the model itself to generate some data, which includes latent spaces, predicted class id, etc. We decide some latent variable to be positive or negative according to the predicted class id. Since there are too many different pairs of anchor-positive and anchor-negative latent variables that can be generated, we only randomly choose some specific number of pairs (we use 1 pair in our experiment) for each epoch.

Note that we don't use memory to save any real samples or latent variables of previous tasks. Instead, we generate data whenever we need. Thus, what the decoder learns in the partitioning phase is the ability to cluster latent variables of the data of the same class closer to each other to form a more compact latent space, as well as pushing latent spaces of data of different classes far away from each other. This ability is saved in the parameters of the decoder, especially in the latent layer. Since the partitioning of latent space actually constrains the direction of how the parameters of the decoder are updated, our method can be considered as a regularization method.

4 EXPERIMENTAL RESULTS

4.1 SETUP AND EVALUATION

We compare the results of our method and other baseline models under similar settings for each method. We evaluate our method and other methods on some datasets, including gray-scale datasets like MNIST, FashionMNIST, and a colorful dataset, CelebA ((Liu et al., 2015)), which is a relatively more complicated dataset.

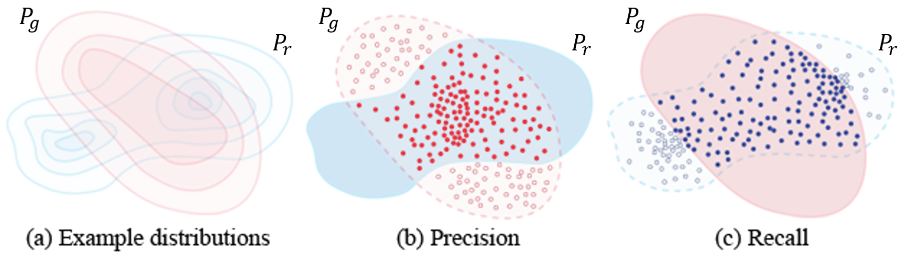


Figure 4: Schematic diagram: Precision and recall for generative models.

Table 1: Results of our method and other baseline methods on MNIST and FashionMNIST.

	MNIST			FashionMNIST		
	FID	Precision	Recall	FID	Precision	Recall
Zenke et al. (2017)	153	0.75	0.76	140	0.21	0.19
Kirkpatrick et al. (2017)	120	0.79	0.38	137	0.24	0.22
Shin et al. (2017)	254	0.70	0.65	133	0.35	0.43
Nguyen et al. (2017)	127	0.78	0.80	138	0.21	0.20
Von Oswald et al. (2019)	148	0.78	0.75	155	0.35	0.21
Rao et al. (2019)	181	0.84	0.74	**83**	0.46	0.56
Ramapuram et al. (2020)	224	0.63	0.73	201	0.09	0.49
Deja et al. (2021)	41	0.92	**0.96**	87	0.56	**0.65**
Ours	**19**	**0.97**	**0.96**	**83**	**0.61**	**0.65**

To evaluate the quality of generated images quantitatively, we use metrics such as FID (Heusel et al., 2017), precision and recall (Kynkäänniemi et al. (2019), schematic diagram is showed in Figure 4). FID is used to assess the quality and diversity of generated images and the lower means the better results. Precision and recall are used to evaluate the ability of the model to capture the distribution of real data. Specifically, we denote the distribution of generated data and real data as P_g and P_r[reference], respectively, then $Precision = \frac{P_g \bigwedge P_r}{P_g}$ means the proportion of generated data of P_g that fall in P_r (the red dots in figure (b)) to the whole P_g, i.e. how much the model predicts the real distribution wrongly; and $Recall = \frac{P_g \bigwedge P_r}{P_r}$ means the proportion of generated data of P_g that fall in P_r (the blue dots in figure(c)) to the whole P_r, i.e. how much the model can capture the real distribution.

4.2 Results on gray-scale datasets

Table 1 shows the results on gray-scale datasets compared to other methods. Our method achieved better results on FashionMNIST and especially MNIST, which shows the effectiveness of triplet learning. We think that the reason is that the data distribution of MNIST and FashionMNIST are not so complex. What's more, since the predicted class id, which we use to decide anchor, positive and negative latent variables, is also generated by the model itself, the model itself can affect its learning outcome in later training. As a dataset that consists of simpler images (i.e. handwritten digits) than FashionMNIST, which consists of clothes and pants, etc., MNIST is easier for the model to learn. Hence, triplet loss works more effectively on MNIST than FashionMNIST.

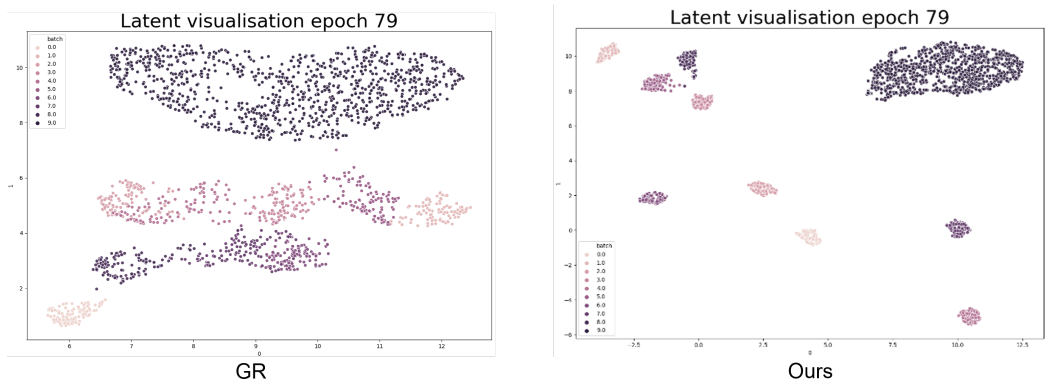


Figure 5: Comparison of visualised latent space trained on MNIST between our method and Generative Replay (GR). It is obvious that our method achieved better partitioning result due to triplet loss.

Table 2: Results of our method, Generative Replay and independent models on CelebA. Here, the independent models mean the mode where we train a model for each single task.

	CelebA		
	FID	Precision	Recall
Jointly training	88	0.35	0.30
Independent models	103	0.31	0.21
GR	104	0.19	0.13
Ours	**90**	**0.322**	**0.272**

Table 3: Results of our method and other baseline methods on CelebA, when trained after different epochs and with different margins.

Epoch	Metric	Margin			
		1	3	5	10
100	FID	89.714	89.714	89.714	102.21
	Precision	0.322	0.340	0.340	0.285
	Recall	0.272	0.282	0.267	0.210
150	FID	86.325	86.340	86.339	99.665
	Precision	0.42	0.44	0.40	0.315
	Recall	0.35	0.36	0.35	0.231

Another discovery is that we achieve much higher precision and recall on both datasets, which indicates that our method has a better ability to capture the data distribution.

4.3 Results on more complex dataset

Image generation under continual learning setup has always been a challenge on complex datasets. Therefore, we also did some experiments on CelebA. The quantitative results are shown in Table 2. Independent models mean the mode where there is a single VAE model trained for each task, i.e. knowledge is not transferable between tasks. Except for the upper bound shown by jointly training mode, our method achieves the best performance on all the metrics, showing the potential of triplet in latent space partitioning even on complex dataset. As qualitative results, comparison of the visualization of latent spaces between GR and our method are shown in Figure 5.

4.4 Experiment of how margin effects performance

Distance in embedding space is an abstract concept, so we are curious about what value of $margin$ is better for partitioning. Since our model has already achieved relatively better results on the gray-scale datasets, we run some experiments on CelebA, to see the effect of $margin$. Table 3 shows the results as follows:

We can find out that the margin of 1, 3 and 5 almost work the same, but performance drop is detected when margin is set to be 10. We infer that the strength of partitioning may increase non-linearly, or even exponentially with $margin$.

5 Conclusion

We have applied triplet loss to do latent space partitioning straightforwardly to solve the forgetting problem in continual learning setup, and evaluated different hyperparameters of triplet loss to see different partition strength. The results show that with the help of triplet loss, we can achieve better result on different dataset when evaluated with different metrics.

One problem of our method is that we neglect the similarity of data from different classes. For example, man, woman, kid, old people belong to different classes, but they all share many similarities of human: two arms, two legs, etc. Total partitioning of latent space means no interference between knowledge of data of different classes, but it may be not the best circumstance when there is transferable knowledge between different classes. Then the intuitional problem comes: will it be better if the latent spaces of data of different classes that share some similarity overlap with each other? Contrary can check for this hypothesis by using a similarity regularizer into the triplet loss to control the strength of partitioning, which we leave as a future work.

REFERENCES

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 139–154, 2018.

Craig Atkinson, Brendan McCane, Lech Szymanski, and Anthony Robins. Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks. *arXiv preprint arXiv:1802.03875*, 2018.

Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co2l: Contrastive continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9516–9525, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pp. 539–546. IEEE, 2005.

Matthias De Lange and Tinne Tuytelaars. Continual prototype evolution: Learning online from non-stationary data streams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8250–8259, 2021.

Kamil Deja, Paweł Wawrzyński, Daniel Marczak, Wojciech Masarczyk, and Tomasz Trzciński. Multiband vae: Latent space partitioning for knowledge consolidation in continual learning. *arXiv preprint arXiv:2106.12196*, 2021.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Jongheon Jeong and Jinwoo Shin. Training gans with stronger augmentations via contrastive discriminator. *arXiv preprint arXiv:2103.09742*, 2021.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *Advances in Neural Information Processing Systems*, 32, 2019.

Frantzeska Lavda, Jason Ramapuram, Magda Gregorova, and Alexandros Kalousis. Continual classification learning using generative models. *arXiv preprint arXiv:1810.10612*, 2018.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10203–10209. IEEE, 2020.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Jason Ramapuram, Magda Gregorova, and Alexandros Kalousis. Lifelong generative modeling. *Neurocomputing*, 404:381–400, 2020.

Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. Continual unsupervised representation learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

Johannes Von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995. PMLR, 2017.