DDGR: Continual Learning with Deep Diffusion-based Generative Replay

Rui Gao [1] **Weiwei Liu** [1]

Abstract

Popular deep-learning models in the field of image classification suffer from catastrophic forgetting—models will forget previously acquired skills when learning new ones. Generative replay (GR), which typically consists of a generator and a classifier, is an efficient way to mitigate catastrophic forgetting. However, conventional GR methods only focus on a single instruction relationship (generator-to-classifier), where the generator synthesizes samples for previous tasks to instruct the training of the classifier, while ignoring the ways in which the classifier can benefit the generator. In addition, most generative replay methods typically reuse the generated samples to update the generator, which causes the samples regenerated by the generator deviating from the distribution of previous tasks. To overcome these two issues, we propose a novel approach, called deep diffusion-based generative replay (DDGR), which adopts a diffusion model as the generator and calculates an *instruction-operator* through the classifier to instruct the generation of samples. Extensive experiments in class incremental (CI) and class incremental with repetition (CIR) settings demonstrate the advantages of DDGR. Our code is available at https://github.com/xiaocangshengGR/DDGR.

1. Introduction

In the digital world, the number of new tasks is increasing overwhelmingly as huge amounts of data are produced (Liu & Tsang, 2015; Liu et al., 2017; Liu & Tsang, 2017; Liu et al., 2019). This continuous generation of emerging learning tasks requires learning systems to adapt quickly to changing environments. Recently, many studies have shown that existing standard deep learning methods quickly forget their previously acquired experiences when learning new tasks (Kirkpatrick et al., 2017). This phenomenon, referred to as catastrophic forgetting (Kumaran et al., 2016), represents a significant challenge for scenarios in which tasks are learned in sequence and previous training data cannot be obtained.

Continual learning (CL)—also referred to as lifelong learning (Chen & Liu, 2018), sequential learning (Aljundi et al., 2019), and incremental learning (Aljundi et al., 2018)—has emerged as an efficient way to mitigate catastrophic forgetting in deep learning models. Classical CL works have explored a range of approaches to help the model remember previous knowledge. Among them, the generative replay (GR) model is one of the most popular methods due to its promising experimental results (Hayes et al., 2021). In the field of image classification, GR methods usually consist of two components: a generator and a classifier. The generator aims to generate or synthesize samples containing previous task knowledge and feeds these samples to the classifier. The classifier then trains with new data as well as generated samples to mitigate catastrophic forgetting. For example, Kemker & Kanan (2018) propose FearNet, which is a generator that employs a brain-inspired dual-memory system to instruct the classifier training; Wu et al. (2018) propose MeRGANs, which can generate images without forgetting; Ye & Bors (2020) propose L-VAEGAN, which induces generative replay samples and learns informative latent representations.

However, most GR methods suffer from two disadvantages: the adoption of instruction in a single direction (generator-to-classifier) and the reuse of generated samples. In CL, the classifier, which is trained on new data and previous samples, is able to help the generator adjust to the changing tasks. In contrast, most GR methods ignore the benefit offered by the classifier to the generator. Moreover, reusing generated samples may produce low-quality samples for previous tasks. For example, DGR (Shin et al., 2017) trains a new generator on a mixed data distribution of new samples and reused samples. Figure 1 shows that DGR generates fuzzy samples for previous tasks. This paper proposes to use a diffusion probabilistic model (referred to here for brevity as a diffusion model) (Sohl-Dickstein et al., 2015; Ho et al., 2020) to address these two issues.

[1]School of Computer Science, National Engineering Research Center for Multimedia Software, Institute of Artificial Intelligence and Hubei Key Laboratory of Multimedia and Network Communication Engineering, Wuhan University, Wuhan, China. Correspondence to: Weiwei Liu <liuweiwei863@gmail.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

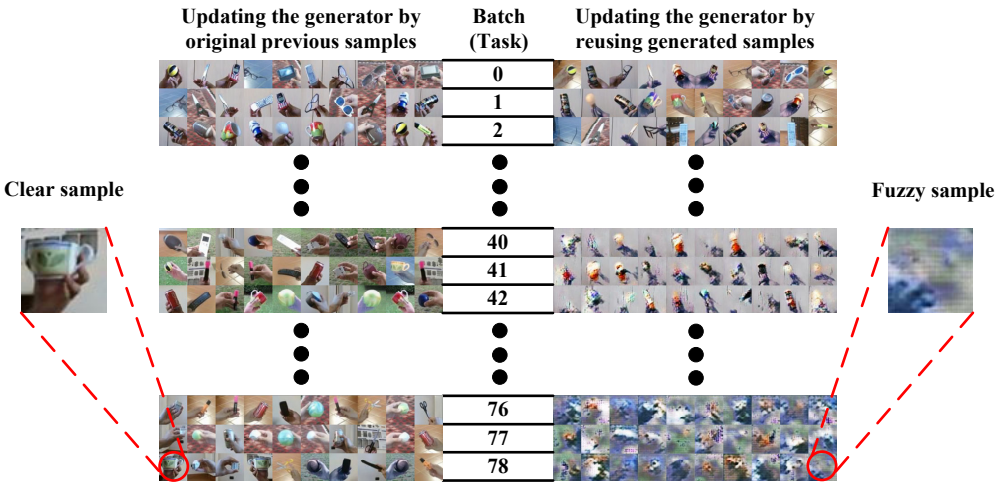


Figure 1. We use DGR as an example to demonstrate that reusing generated samples will lead to the generation of low-quality samples for previous tasks. We conduct experiments on CORe50. Further details of these experiments are provided in Section 3.2.

Diffusion models are motivated by non-equilibrium thermodynamics (Sohl-Dickstein et al., 2015), and are proven to be able to synthesize high-quality images or audio (Ho et al., 2020; Chen et al., 2021; Ho et al., 2022a; Kong et al., 2021). As illustrated in Figure 2, a diffusion model maintains a Markov chain of diffusion steps to gradually add Gaussian noise to data, and learns to reverse the diffusion process in order to generate the desired samples from the noise. Different from VAE (Kingma & Welling, 2014) or GAN (Goodfellow et al., 2014), diffusion models are learned according to a fixed procedure and constructed by latent high-dimensional variables, which are scaled to be the same size as the original data.

In this paper, we propose a novel approach, called deep diffusion-based generative replay (DDGR), which uses a diffusion model as the generator to synthesize samples for previous tasks. DDGR concentrates on a bi-directional instructional relationship between the generator and the classifier. When learning new tasks, DDGR uses the classifier, which is pretrained on previous tasks, to instruct the sampling process of the diffusion model at each time step. New data is combined with the generated samples and used to train the classifier. The diffusion model is also trained on the combined data to enable it to adjust to the changing tasks. Meanwhile, the instruction provided by the classifier to the diffusion model guarantees a high quality of synthetic samples that will not decrease as the tasks change. Our main contributions can be summarized as follows:

- We focus on sampling process of a diffusion model and explore how this process might be instructed by a pretrained classifier. Specifically, we calculate *instruction-operator* through classifier at each time step of diffusion model to guide the generation of samples.
- The novel DDGR is proposed based on a diffusion model, where the classifier uses the *instruction-operator* to instruct the sampling process of the diffusion model. Benefiting from the *instruction-operator*, DDGR significantly improves the quality of generated samples for previous tasks.
- Extensive experimental results under class incremental (CI) and class incremental with repetition (CIR) settings demonstrate the advantages of DDGR.

2. Related Work

Continual learning. Existing researches in CL mainly focus on how to solve the catastrophic forgetting. Some surveys (Parisi et al., 2019; Lesort et al., 2019; Pfülb & Gepperth, 2019; Farquhar & Gal, 2018) have shown that existing CL works can be broadly divided into three categories: (i) regularization-based methods, (ii) parameter isolation methods and (ii) replay methods.

Classical regularization-based methods add an extra regularization term to the loss function. These methods penalize changes to important parameters of model for previous tasks (Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018; Lee et al., 2017). There are also other regularization-based methods that are based on functional regularization (Pan et al., 2020; Benjamin et al., 2019), uncertainty regularization (Ahn et al., 2019), classifier projection regularization (Cha et al., 2021) and node importance (Jung et al., 2020).

Parameter isolation methods dedicate different model parameters to each task to prevent any possible forgetting.

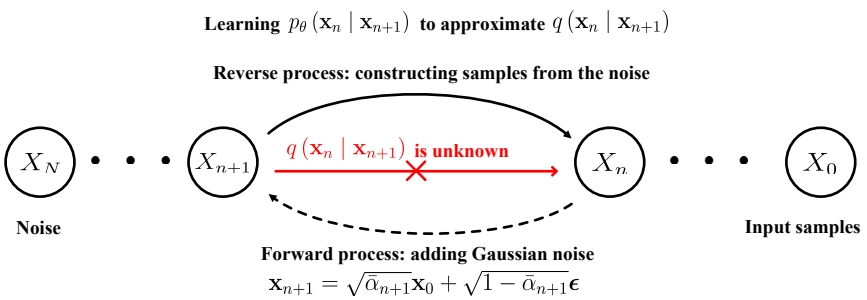


Figure 2. An illustration of diffusion models. Here, we present a DDPM structure (Ho et al., 2020) upon which our work is based. DDPM is a typical diffusion model based on (i) a forward process, in which Gaussian noise is gradually added to the input samples, and (ii) a reverse process, in which the diffusion model aims to recover the original input samples from the noisy data.

When no architectural size constraints apply, it is possible to grow new branches for new tasks while freezing previous task parameters (Xu & Zhu, 2018; Mallya & Lazebnik, 2018; Serrà et al., 2018).

Replay methods generally have two ways to replay in artificial neural networks: partial replay and generative replay (GR)(Hayes et al., 2021). Partial replay stores either all or a subset of previously learned inputs in a replay buffer, and mixes these inputs with new samples to train the classifier (Rebuffi et al., 2017; Lopez-Paz & Ranzato, 2017; Rolnick et al., 2019; Ayub & Wagner, 2021). In contrast to storing previous examples, GR generates synthetic samples for previous tasks. Classical GR methods use VAE or GAN as the generator for CL (Wu et al., 2018; Achille et al., 2018; Zhai et al., 2019; Wu et al., 2019; Ramapuram et al., 2020; Mundt et al., 2022). For example, DGR (Shin et al., 2017) uses GAN to to generate previous samples for data replaying. Wu et al. (2018) propose MeRGANs to solve the problem of forgetting in generative models (GAN in particular), and achieve great performance in CL. In addition, Liu et al. (2020) propose a mnemonics training framework to generate exemplars which are optimizable. Most of these methods perform a single instruction and reuse the generated samples, which we try to solve in this paper. In addition, graph-based replay (Tang & Matteson, 2021) and gradient projection memory (Saha et al., 2021) have also been proven to be efficient.

Diffusion model. A diffusion model is a parameterized Markov chain trained using variational inference to produce samples matching the data after finite time. Diffusion models are based on two components: (i) a forward diffusion stage, in which the input data is gradually perturbed over several steps by adding Gaussian noise; (ii) a reverse diffusion stage, in which a generative model learns to gradually reverse the diffusion process to recover the input data from the diffused (noisy) data. Croitoru et al. (2022) provide a categorization of diffusion models based on the denoising condition: unconditional image generation and conditional image generation. Unconditional image generation do not require supervision signals, being completely unsupervised. For example, denoising diffusion probabilistic model (DDPM) (Ho et al., 2020), which learns the reverse process by estimating the noise in the image at each step, is a typical unconditional diffusion model; On top of DDPM, Nichol & Dhariwal (2021) propose a unconditional diffusion model which shows that it is required to learn the variance in terms of log-likelihood; the work of Nachmani et al. (2021) is also unconditional where they replace the Gaussian noise distributions with two other distributions, a mixture of two Gaussians and the Gamma distribution. Conditional image generation is commonly based on various source signals, in most cases some class labels being used. For example, Pandey et al. (2021) build a generator-refiner framework, where the generator is a VAE and the refiner is a DDPM conditioned by the output of the VAE; Ho & Salimans (2022) introduce a guidance method whose idea is based on the implicit classifier derived from the Bayes rule; Singh et al. (2022) propose a novel method for conditional image generation which conditions the noise signal (from where the sampling starts) instead of conditioning the signal throughout the sampling process. More details can be found in the work of Croitoru et al. (2022). Recently, diffusion models have shown their exquisite potential in the field of computer vision (Dhariwal & Nichol, 2021; Ho et al., 2022b; Li et al., 2022; Luo & Hu, 2021), audio processing (Chen et al., 2021; Popov et al., 2021; Kim et al., 2022), and AI for science (Luo et al., 2021; Xu et al., 2022; Jing et al., 2022). In this paper, we show the advantages of diffusion models in CL.

3. Preliminaries

3.1. Denoising diffusion probabilistic model

Our work is based on DDPM (Ho et al., 2020) which is a typical diffusion model architecture. As shown in Figure 2, DDPM consists of a forward and a reverse processes.

Forward process. Given a data point sampled from a real data distribution $\mathbf{x}_0 \sim q(\mathbf{x_0})$, the forward process is fixed to a Markov chain, which gradually adds Gaussian noise to the data according to a variance schedule $\{\beta_n \in (0,1)\}_{n=1}^N$:

$$q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right) = \prod_{n=1}^{N} q\left(\mathbf{x}_n \mid \mathbf{x}_{n-1}\right) \tag{1}$$

$$q\left(\mathbf{x}_n \mid \mathbf{x}_{n-1}\right) = \mathcal{N}\left(\mathbf{x}_n; \sqrt{1-\beta_n}\mathbf{x}_{n-1}, \beta_n \mathbf{I}\right) \tag{2}$$

where $\sqrt{1-\beta_n}\mathbf{x}_{n-1}$ and $\beta_n\mathbf{I}$ are the mean and variance of the Gaussian distribution respectively, while $\mathbf{x}_n$ is the noisy sample at time step n. It is noteworthy that the mean, variance, and noisy sample have the same dimensionality as $\mathbf{x}_0$. This process produces a sequence of noisy samples $\mathbf{x}_1, ..., \mathbf{x}_N$ in N steps. Based on reparameterization trick, $\mathbf{x}_n$ at any time step n can be derived by $\mathbf{x}_0$ as follows:

$$\mathbf{x}_n = \sqrt{\bar{\alpha}_n}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_n}\boldsymbol{\epsilon} \tag{3}$$

$$q\left(\mathbf{x}_n \mid \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_n; \sqrt{\bar{\alpha}_n}\mathbf{x}_0, \left(1-\bar{\alpha}_n\right)\mathbf{I}\right) \tag{4}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\alpha_n = 1 - \beta_n$ and $\bar{\alpha}_n = \prod_{i=1}^{n} \alpha_i$ (see Appendix B for details of Equations (3) and (4)).

Reverse process. If we can reverse the above process and generate samples from $q\left(\mathbf{x}_{n-1} \mid \mathbf{x}_n\right)$, it is possible to construct the original input from the Gaussian noise. However, the conditional probability $q\left(\mathbf{x}_{n-1} \mid \mathbf{x}_n\right)$ is unknown. An alternative is to train a model with parameter θ to learn a conditional probability $p_\theta\left(\mathbf{x}_{n-1} \mid \mathbf{x}_n\right)$ that approximates $q\left(\mathbf{x}_{n-1} \mid \mathbf{x}_n\right)$. The reverse process can be expressed as follows:

$$p_\theta\left(\mathbf{x}_{0:N}\right) = p\left(\mathbf{x}_N\right) \prod_{n=1}^{N} p_\theta\left(\mathbf{x}_{n-1} \mid \mathbf{x}_n\right) \tag{5}$$

$$p_\theta\left(\mathbf{x}_{n-1} \mid \mathbf{x}_n\right) = \mathcal{N}\left(\mathbf{x}_{n-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_n, n\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_n, n\right)\right) \tag{6}$$

Training objective. The setup above is similar to VAE. To formulate this minimization problem, a variational lower bound is used to optimize the negative log-likelihood:

$$\begin{aligned} -\log p_\theta\left(\mathbf{x}_0\right) &\leq -\log p_\theta\left(\mathbf{x}_0\right) \\ &\quad + D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)\right) \\ &= \mathbb{E}_{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)}\right] \end{aligned} \tag{7}$$

$$-\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0) \leq \mathbb{E}_{q(\mathbf{x}_{0:N})}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)}\right] =: L_{LB} \tag{8}$$

Sohl-Dickstein et al. (2015) split the objective L_{LB} into several KL-divergence terms, where each term in the objective is analytically computable:

$$\begin{aligned} L_{LB} &= \mathbb{E}_{q(\mathbf{x}_{0:N})}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)}\right] \\ &= \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_N \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_N\right)\right)}_{L_N} \\ &\quad + \sum_{n \geq 1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)\right)}_{L_n} \\ &\quad \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]. \end{aligned} \tag{9}$$

DDPM fixes β_n to constants, and L_N can be ignored because it is also constant. L_0 is modelled by a separate discrete decoder derived from $\mathcal{N}\left(\mathbf{x}_0; \boldsymbol{\mu}_\theta\left(\mathbf{x}_1, 1\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_1, 1\right)\right)$. As for L_n, it is noteworthy that $q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right)$ of L_n is traceable:

$$q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_n; \tilde{\boldsymbol{\mu}}_{n+1}, \tilde{\boldsymbol{\Sigma}}_{n+1}\right) \tag{10}$$

$$\tilde{\boldsymbol{\mu}}_{n+1} = \frac{1}{\sqrt{\alpha_{n+1}}}\left(\mathbf{x}_{n+1} - \frac{1-\alpha_{n+1}}{\sqrt{1-\bar{\alpha}_{n+1}}}\boldsymbol{\epsilon}\right) \tag{11}$$

$$\tilde{\boldsymbol{\Sigma}}_{n+1} = \frac{1-\bar{\alpha}_n}{1-\bar{\alpha}_{n+1}} \cdot \beta_{n+1}\mathbf{I} \tag{12}$$

In DDPM, $\tilde{\boldsymbol{\Sigma}}_{n+1}$ are set to $\sigma_{n+1}^2\mathbf{I}$, which are time dependent constants. As for $\tilde{\boldsymbol{\mu}}_{n+1}$, it is only necessary to train $\boldsymbol{\epsilon}_\theta(\mathbf{x}_{n+1}, n+1)$ to predict $\boldsymbol{\epsilon}$, because $\mathbf{x}_{n+1}$ can be derived at training time. Therefore, the objective is:

$$L_n = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[\omega \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_{n+1}, n+1)\right\|^2\right] + C \tag{13}$$

where $\omega = \frac{(1-\alpha_{n+1})^2}{2\sigma_{n+1}^2\alpha_{n+1}(1-\bar{\alpha}_{n+1})}$ is the weighting term, while C is a constant term that does not depend on θ. $\boldsymbol{\epsilon}_\theta(\mathbf{x}_{n+1}, n+1)$ is a model with parameter θ, which takes $\mathbf{x}_{n+1}$ and $n+1$ as inputs. Ho et al. (2020) find that it is better to train the diffusion model without the weighting term. The final objective can be expressed as follows:

$$\mathbb{E}_{n \sim [1, N-1], \mathbf{x}_0, \boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_{n+1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{n+1}}\boldsymbol{\epsilon}, n+1\right)\right\|^2\right] \tag{14}$$

where $\mathbf{x}_{n+1}$ in Equation (13) is replaced according to Equation (3). Further details regarding the training objective can be found in Appendix C.

Sampling. After training the model $\boldsymbol{\epsilon}_\theta(\mathbf{x}_{n+1}, n+1)$, the conditional probability, the mean and variance of which are $\frac{1}{\sqrt{\alpha_{n+1}}}\left(\mathbf{x}_{n+1} - \frac{1-\alpha_{n+1}}{\sqrt{1-\bar{\alpha}_{n+1}}}\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_{n+1}, n+1\right)\right)$ and $\sigma_{n+1}^2\mathbf{I}$ respectively, is known. Starting at $\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the complete sampling process can be run according to the following equation:

$$\mathbf{x}_n = \frac{1}{\sqrt{\alpha_{n+1}}}\left(\mathbf{x}_{n+1} - \frac{1-\alpha_{n+1}}{\sqrt{1-\bar{\alpha}_{n+1}}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_{n+1}, n+1)\right) + \sigma_{n+1}\mathbf{z} \tag{15}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $n = N-1, \cdots, 0$.

3.2. Preliminary experiments

To demonstrate that reusing generated samples will lead to low-quality generated samples for previous tasks, we conduct preliminary experiments on CORe50 (Lomonaco & Maltoni, 2017). Details of the setup on CORe50 can be found in Section 5.1. We take DGR as an example and run DGR twice on CORe50. In the first run, we assume that the generator has access to the previous task data, and use the original data of the previous tasks to update the generator. In the second run, we reuse the generated samples to update the DGR generator. We investigate the generated samples at each task over the two runs. Partial samples are presented in Figure 1. We find that the samples in the second run (in which the generated samples are reused) become blurred as new tasks are learned. In contrast, the samples in the first run can still be clearly distinguished as new tasks are learned. These results indicate that reusing the generated samples will decrease the quality of the generated samples when learning new tasks.

4. Deep diffusion-based generative replay

In this section, we present our proposed method—DDGR. We adopt DDPM as the generator. Moreover, we utilize a bi-direction relationship between the generator and the classifier. Similar to most GR methods, DDGR uses the generator to generate samples for use in training the classifier for previous tasks. In addition, DDGR also uses the classifier, which is pretrained on previous tasks, to instruct the generator to synthesize high-quality samples of previous tasks. Specifically, we calculate an *instruction-operator*, which affects the noisy sample, through the classifier at each time step of the diffusion model.

4.1. Instruction-operator

In classifier-to-generator instruction, we focus primarily on the sampling process of the diffusion model. Dhariwal & Nichol (2021) inspire us to achieve this by a classifier. We assume that the classifier $\boldsymbol{f}(x)$ is pretrained on previous sample set $\mathcal{X}$ (drawn from $\mathcal{D}_{pre}$, which is the mixed distribution of previous tasks) with label set $\mathcal{Y} = \{y_1, y_2, \cdots, y_m\}$. To mitigate catastrophic forgetting, the classifier requires high-quality generated samples whose labels belong to $\mathcal{Y}$. Therefore, we need to consider how to condition the sample process of DDPM on a specific label y.

We first condition the entire Markov process of DDPM on label y, and define conditional Markov process as $\hat{q}$ (to distinguish it from unconditional Markov process q, which corresponds to DDPM). The key role of conditional sampling on y is to track the conditional probability $\hat{q}(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y)$. It is noteworthy that we have the following theorem:

Theorem 4.1. *$\hat{q}(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y)$ is proportional to $q(\mathbf{x}_n \mid \mathbf{x}_{n+1})\,\hat{q}(y \mid \mathbf{x}_n)$, where $q(\mathbf{x}_n \mid \mathbf{x}_{n+1})$ is the conditional probability in the reverse process of q, and $\hat{q}(y \mid \mathbf{x}_n)$ is the label distribution given $\mathbf{x}_n$.*

See Appendix D for more details. We assume that $\hat{q}(y \mid \mathbf{x}_n)$ in Theorem 4.1 is approximated by $p_\zeta(y \mid \mathbf{x}_n)$ with parameter ζ. As introduced in Section 3.1, we use $p_\theta(\mathbf{x}_n \mid \mathbf{x}_{n+1})$ to approximate $q(\mathbf{x}_n \mid \mathbf{x}_{n+1})$ after training the diffusion model. By plugging $p_\zeta(y \mid \mathbf{x}_n)$ and $p_\theta(\mathbf{x}_n \mid \mathbf{x}_{n+1})$ into Theorem 4.1, we can derive the following:

$$\hat{q}(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y) = \mathbf{C}_{nor} p_\theta(\mathbf{x}_n \mid \mathbf{x}_{n+1})\, p_\zeta(y \mid \mathbf{x}_n) \quad (16)$$

where $\mathbf{C}_{nor}$ is a normalizing constant term. Recalling that the conditional probability in sampling process is $p_\theta(\mathbf{x}_n \mid \mathbf{x}_{n+1}) = \mathcal{N}\left(\mathbf{x}_n; \boldsymbol{\mu}_\theta(\mathbf{x}_{n+1}, n+1), \sigma_{n+1}^2 \mathbf{I}\right)$, according to the Gaussian density function, we can then derive $\log p_\theta(\mathbf{x}_n \mid \mathbf{x}_{n+1})$ as follows:

$$\begin{aligned}\log p_\theta(\mathbf{x}_n \mid \mathbf{x}_{n+1}) &= \mathbf{C} - \frac{(\mathbf{x}_n - \boldsymbol{\mu}_\theta)^2}{2\sigma_{n+1}^2 \mathbf{I}} \\ &= \mathbf{C} - (\mathbf{x}_n - \boldsymbol{\mu}_\theta)^\top (2\sigma_{n+1}^2 \mathbf{I})^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_\theta)\end{aligned} \quad (17)$$

where $\mathbf{C}$ is a constant term independent from $\mathbf{x}_n$, while $\boldsymbol{\mu}_\theta$ represents $\boldsymbol{\mu}_\theta(\mathbf{x}_{n+1}, n+1)$ for convenience. We also use $\mathbf{C}$ to represent all constant terms that are independent from $\mathbf{x}_n$ in the following steps for convenience. As for $\log p_\zeta(y \mid \mathbf{x}_n)$, we use the Taylor expansion of $\log p_\zeta(y \mid \mathbf{x}_n)$ at $\mathbf{x}_n = \boldsymbol{\mu}_\theta$ as an approximation of itself.

$$\begin{aligned}\log p_\zeta(y \mid \mathbf{x}_n) \approx{}& \log p_\zeta(y \mid \mathbf{x}_n)|_{\mathbf{x}_n = \boldsymbol{\mu}_\theta} + \\ & (\mathbf{x}_n - \boldsymbol{\mu}_\theta) \nabla_{\mathbf{x}_n} \log p_\zeta(y|\mathbf{x}_n)|_{\mathbf{x}_n = \boldsymbol{\mu}_\theta} \\ ={}& \mathbf{C} + (\mathbf{x}_n - \boldsymbol{\mu}_\theta)\,\mathbf{G}\end{aligned} \quad (18)$$

where $\mathbf{G} = \nabla_{\mathbf{x}_n} \log p_\zeta(y|\mathbf{x}_n)|_{\mathbf{x}_n = \boldsymbol{\mu}_\theta}$. We can derive $\log \hat{q}(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y)$ by Equations (16), (17) and (18):

$$\log \hat{q}(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y) \quad (19)$$

$$= \log(p_0(\mathbf{x}_n|\mathbf{x}_{n+1}) p_0(g|\mathbf{x}_n)) + \mathbf{C} \quad (20)$$

$$\approx -\frac{(\mathbf{x}_n - \boldsymbol{\mu}_\theta)^2}{2\sigma_{n+1}^2 \mathbf{I}} + (\mathbf{x}_n - \boldsymbol{\mu}_\theta)\,\mathbf{G} + \mathbf{C} \quad (21)$$

$$= -\frac{\left(\mathbf{x}_n - \boldsymbol{\mu}_\theta - \sigma_{n+1}^2 \mathbf{G}\right)^2}{2\sigma_{n+1}^2 \mathbf{I}} + \frac{\sigma_{n+1}^2}{2}\mathbf{G}^2 + \mathbf{C} \quad (22)$$

$$= -\frac{\left(\mathbf{x}_n - \boldsymbol{\mu}_\theta - \sigma_{n+1}^2 \mathbf{G}\right)^2}{2\sigma_{n+1}^2 \mathbf{I}} + \mathbf{C} \quad (23)$$

$$= \mathbf{C} - \left(\mathbf{x}_n - \boldsymbol{\mu}_\theta - \sigma_{n+1}^2 \mathbf{G}\right)^\top (2\sigma_{n+1}^2 \mathbf{I})^{-1} \left(\mathbf{x}_n - \boldsymbol{\mu}_\theta - \sigma_{n+1}^2 \mathbf{G}\right) \quad (24)$$

where the constant term $\mathbf{C}$ can be ignored due to the normalizing constant $\mathbf{C}_{nor}$ in Equation (16). Recalling the Gaussian density distribution, Equation (24) means that $\hat{q}(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y)$ can be approximated by

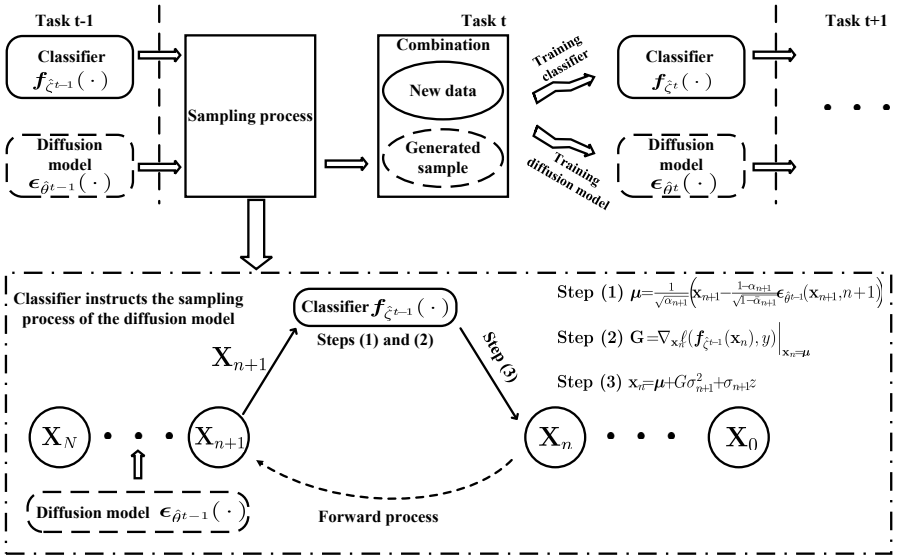


Figure 3. An illustration of how the *instruction-operator* works in the sampling process of the diffusion model. At each time step, we perform steps (1) and (2) to calculate an *instruction-operator* through the classifier. We plug the *instruction-operator* into the sample transition at step (3) and derive the next noisy sample $\mathbf{x}_n$.

$\mathcal{N}\left(\mathbf{x}_n; \boldsymbol{\mu}_\theta + \sigma^2_{n+1}\mathbf{G}, \sigma^2_{n+1}\mathbf{I}\right)$. The condition of the sampling process given label y is as presented below:

$$\mathbf{x}_n = \boldsymbol{\mu}_\theta + \sigma^2_{n+1}\mathbf{G} + \sigma_{n+1}\mathbf{z} \tag{25}$$

where $\boldsymbol{\mu}_\theta = \frac{1}{\sqrt{\alpha_{n+1}}}\left(\mathbf{x}_{n+1} - \frac{1-\alpha_{n+1}}{\sqrt{1-\bar{\alpha}_{n+1}}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_{n+1}, n+1)\right)$ and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Let us now turn our attention to the operator $\mathbf{G} = \nabla_{\mathbf{x}_n} \log p_\zeta\left(y \mid \mathbf{x}_n\right)\big|_{\mathbf{x}_n=\boldsymbol{\mu}_\theta}$. In the operator, $p_\zeta\left(y \mid \mathbf{x}_n\right)$ is essentially a classifier that takes $\mathbf{x}_n$ as input and approximates $\hat{q}\left(y \mid \mathbf{x}_n\right)$ with parameter ζ. We assume that $\mathbf{x}_n$ is able to be drawn from $\mathcal{D}_{pre}$, which indicates that $p_\zeta\left(y \mid \mathbf{x}_n\right)$ and the pretrained classifier $\boldsymbol{f}(\cdot)$ have similar input domains. Therefore, we regard the pretrained classifier $\boldsymbol{f}(\cdot)$ as $p_\zeta\left(y \mid \mathbf{x}_n\right)$; this means that we can utilize the pretrained classifier $\boldsymbol{f}(\cdot)$ to instruct the sampling process of the diffusion model by the operator, as follows:

$$\mathbf{G} = \nabla_{\mathbf{x}_n} \log p_\zeta(y|\mathbf{x}_n)\big|_{\mathbf{x}_n=\boldsymbol{\mu}_\theta} = \nabla_{\mathbf{x}_n}\ell(\boldsymbol{f}_\zeta(\mathbf{x}_n), y)\big|_{\mathbf{x}_n=\boldsymbol{\mu}_\theta} \tag{26}$$

where $\ell(\cdot)$ is the loss function. We refer to this operator (26) as the *instruction-operator*, which plays an important role in classifier-to-generator instruction. An illustration of how the *instruction-operator* works in the sample process of the diffusion model can be found in Figure 3. When the diffusion model generates samples from Gaussian noise, we calculate the *instruction-operator* through the pretrained classifier $\boldsymbol{f}_\zeta(\cdot)$ and plug it into the sample transition (i.e. Equation (25)). This *instruction-operator* affects the noisy sample at each time step, enabling the diffusion model to generate the desirable samples for the classifiers.

4.2. Generative replay with *instruction-operator*

Overview. Here, we provide an overview of our proposed method DDGR in Algorithm 1. DDGR uses DDPM as the generator and adopts a bi-directional instruction relationship between the generator and the classifier. It is noteworthy that the previous classifier $\boldsymbol{f}_{\hat{\zeta}^{t-1}}$ was trained in sequence on tasks $1, 2, \ldots, t-1$, which indicates that $\boldsymbol{f}_{\hat{\zeta}^{t-1}}$ can be regarded as the pretrained classifier $\boldsymbol{f}_\zeta(\cdot)$. When learning a new task, DDGR first uses a classifier pretrained on previous tasks to instruct the diffusion model to generate samples with previous labels (Line 4 of Algorithm 1). Specifically, we calculate an *instruction-operator* through the classifier at each time step of the diffusion model and plug it into the sample transition (details provided in Algorithm 2). After several iterations, the diffusion model generates the desirable samples with previous labels for the classifier. DDGR then trains the classifier with a combination of generated samples and new data (Line 9 of Algorithm 1). Finally, DDGR updates the diffusion model according to Equation (14) (Lines 10 to 17 of Algorithm 1).

Generated sample reuse issue. As shown in line 12 of Algorithm 1, DDGR also reuses the generated samples to update the diffusion model as the tasks change. However, benefiting from the instruction of the classifier, DDGR significantly improves the quality of the samples of previous tasks produced by the generator. When learning a new task, the classifier is a pretrained model, which means that the classifier contains rich knowledge of previous tasks. The *instruction-operator* can be intuitively regarded as a type of

Algorithm 1 Deep Diffusion-based Generative Replay

Input: $(\tau_1, \tau_2, ..., \tau_T)$: all training tasks; T: total number of tasks; S^t: the sample set of task τ_t; $\boldsymbol{\epsilon}_\theta$: the training model of the diffusion model with parameter θ; N: number of diffusion model steps; NS: the number of generated samples; f_ζ: the classifier with parameter ζ.

$\mathcal{Y} = \{\}$
for $t = 1, \ldots, T$ **do**
  if $t \geq 2$ **then**
    $\overline{S^t}$=$InstructionProcess(\boldsymbol{\epsilon}_{\hat{\theta}^{t-1}}, \boldsymbol{f}_{\hat{\zeta}^{t-1}}, N, NS, \mathcal{Y})$
    $\overline{S^t}$=$\overline{S^t} \cup S^t$
  else
    $\overline{S^t}$=S^t
  end if
  $\hat{\zeta}^t = \text{argmin}_\zeta \, \ell(\boldsymbol{f}_\zeta(\overline{S^t}))$
  $\hat{\theta}^t = \hat{\theta}^{t-1}$
  repeat
    $\mathbf{x}_0$ = a random sample in $\overline{S^t}$
    $n \sim \text{Uniform}(\{1, \ldots, N\})$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_n = \sqrt{\bar{\alpha}_n}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_n}\boldsymbol{\epsilon}$ Equation (3)
    Update $\hat{\theta}^t$ by $\nabla_\theta \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_n, n)\|^2$ Equation (14)
  until converged
  Update $\mathcal{Y}$
end for

Output: $\hat{\zeta}^T$

Algorithm 2 *InstructionProcess*

Input: $\boldsymbol{\epsilon}_\theta$: the training model of the diffusion model with parameter θ; f_ζ: the classifier with parameter ζ; N: number of diffusion model steps; NS: the number of generated samples; $\mathcal{Y}$: label set.

$counter = 0$ and $\overline{S^t} = \{\}$
repeat
  y = a random label in $\mathcal{Y}$
  $\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  for $n = N-1, \ldots, 0$ **do**
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\boldsymbol{\mu}$=$\frac{1}{\sqrt{\alpha_{n+1}}}\left(\mathbf{x}_{n+1} - \frac{1-\alpha_{n+1}}{\sqrt{1-\bar{\alpha}_{n+1}}}\boldsymbol{\epsilon}_{\hat{\theta}^{t-1}}(\mathbf{x}_{n+1}, n+1)\right)$
    $\mathbf{G}$ =$\nabla_{\mathbf{x}_n}\ell(\boldsymbol{f}_{\hat{\zeta}^{t-1}}(\mathbf{x}_n), y)\Big|_{\mathbf{x}_n=\boldsymbol{\mu}}$ Equation (26)
    $\mathbf{x}_n = \boldsymbol{\mu} + \sigma_{n+1}^2\mathbf{G} + \sigma_{n+1}\mathbf{z}$ Equation (25)
  end for
  $\overline{S^t} = \overline{S^t} \cup \{(\mathbf{x}_0, y)\}$
  $counter = counter + 1$
until $counter \geq NS$

Output: $\overline{S^t}$

distillation of previous knowledge to classifier. At each time step of the sampling process, DDGR adds the *instruction-operator* to the sample transition, which means that DDGR continuously adds previous knowledge to the noisy sample $\mathbf{x}_n$. This sampling process significantly strengthens the generated samples, which prevents the generated samples from deviating from the distribution of previous tasks. Therefore, DDGR is not impacted by the problems that typically result from generated sample reuse. We provide the samples generated by DDGR at the last three batches of CORe50 (see details of the experimental settings in Section 5.1). The samples generated by DDGR can still be clearly distinguished. Details can be found in Appendix E.

5. Experiments

We conduct extensive experiments to verify the superior performance of our proposed DDGR. In this paper, we consider two scenarios commonly encountered in CL, namely CI (van der Ven & Tolias, 2018; van de Ven & Tolias, 2019) and CIR (Lomonaco & Maltoni, 2017; Cossu et al., 2022).

5.1. Experimental setup

Baselines: In order to verify the superiority of our proposed method, we select eight baselines for comparison: **SI** (Zenke et al., 2017), **MAS** (Aljundi et al., 2018), **EWC** (Kirkpatrick et al., 2017), **IMM** (Lee et al., 2017), **DGR** (Shin et al., 2017), **MeRGAN** (Wu et al., 2018), **PASS** (Zhu et al., 2021) and **Finetuning**. **SI**, **MAS**, **EWC**, and **IMM** can be considered as the prior-focused methods, which estimate a distribution of model parameters as the prior when learning from new data. These methods typically estimate the importance of all neural network parameters, with parameters assumed to be independent to ensure feasibility. **DGR** and **MeRGAN** use GAN to generate previous samples for data replaying. **PASS** is a simple non- exemplar based method. See Appendix A for details.

Datasets: In CI scenario, we conduct experiments on two widely used datasets: CIFAR-100 and ImageNet (Deng et al., 2009). For this scenario, we follow Liu et al. (2020) and utilize a similar division of datasets, which has one initial task and four incremental tasks. For CIFAR-100, initial task consists of 50 random classes, and all incremental tasks have the same number of classes: specifically, the number of classes in one incremental task is set to either 5 or 10. ImageNet contains around 1.3 million samples of 224×224 color images from 1000 classes. In ImageNet, initial task consists of 500 random classes, and the number of classes in one incremental task is set to either 50 or 100. Moreover, in CIR scenario, we use CORe50 (Lomonaco & Maltoni, 2017) to conduct experiments. The data in CORe50 was collected during eleven distinct sessions (eight indoor and three outdoor) characterized by different backgrounds and lighting conditions. The dataset consists of 50 domestic objects belonging to 10 categories. Classification in this paper is

Table 1. All results on two datasets in the CI scenario. We present the final average accuracy A_T and forgetting rate F_T^{avg}.

	$A_T = acc_{T,0:T}$								$F_T^{avg} = \sum_{c \in C_T} F_T^c / \lvert C_T \rvert$							
	CIFAR-100				ImageNet				CIFAR-100				ImageNet			
Method	AlexNet		ResNet		AlexNet		ResNet		AlexNet		ResNet		AlexNet		ResNet	
	NC=5	10	5	10	50	100	50	100	5	10	5	10	50	100	50	100
Finetuning	6.11	5.12	18.08	17.50	5.33	3.24	12.95	10.28	60.45	59.87	61.65	62.79	56.55	57.83	58.58	59.71
SI	16.96	13.57	26.45	23.15	19.38	14.38	28.88	24.38	48.58	50.18	52.27	56.65	41.18	45.94	41.93	44.56
EWC	15.29	9.71	25.49	18.82	15.22	13.03	23.51	22.03	50.38	54.27	52.83	60.47	45.65	47.01	46.98	46.96
MAS	20.13	18.94	29.94	28.28	16.35	14.51	31.25	25.51	45.75	45.37	49.38	51.31	44.85	45.70	39.55	44.34
IMM	11.26	9.87	21.02	19.79	13.68	11.13	23.19	19.73	54.60	54.57	58.12	59.89	46.65	49.31	47.70	49.62
DGR	42.49	38.16	52.96	48.94	43.94	38.81	53.32	47.56	24.08	26.52	26.36	31.14	17.31	22.52	17.96	21.84
MeRGAN	46.03	43.23	57.19	55.69	—	—	—	—	36.95	26.49	20.12	22.55	—	—	—	—
PASS	53.21	48.65	62.30	**60.63**	—	—	—	—	27.35	19.43	16.97	21.21	—	—	—	—
DDGR	**59.20**	**52.22**	**63.40**	60.04	**53.86**	**52.21**	**64.83**	**61.26**	**23.00**	**16.86**	**15.34**	**19.25**	**6.98**	**7.82**	**5.65**	**7.73**

performed at object level (50 classes). We follow Lomonaco & Maltoni (2017), which generates 79 tasks from CORe50. The first task includes ten classes, and each subsequent task contains five classes. See Appendix A for details.

Models: For the classifier, we follow the work of Rebuffi et al. (2017); Liu et al. (2020); Delange et al. (2021) and use a 32-layer ResNet and AlexNet (Krizhevsky et al., 2012) for all experiments. Moreover, following the work of Ho et al. (2020), we use the UNet architecture for the diffusion model. The UNet model uses a stack of residual layers and downsampling convolutions, followed by a stack of residual layers with upsampling colvolutions, with skip connections connecting layers with the same spatial size.

Evaluation: In the CI scenario, we use two evaluation metrics which are similar to the work of Liu et al. (2020); Choi et al. (2021) . The **average accuracy** at task i is defined as $A_i = acc_{i,0:i}$, where the learned model M_i is evaluated on the test dataset $S_{test}^{0:i}$. $S_{test}^{0:i}$ denotes all data (classes) seen so far. The forgetting rate of the previous class c at task i is defined as $F_i^c = acc_{j,c} - acc_{i,c}$, where $acc_{j,c}$ and $acc_{i,c}$ are the accuracies of models M_j and M_i on the test dataset S_{test}^c, respectively. M_j is the first model to see class c, and S_{test}^c is the dataset with only one label c. The **average forgetting rate** at task i is defined as $F_i^{avg} = \sum_{c \in C_i} F_i^c / |C_i|$, where C_i is the set of all seen classes. In the CIR scenario, we follow the work of Lomonaco & Maltoni (2017) and evaluate the learned model M_i by the **average accuracy** $A_i^{FTS} = acc_{i,FTS}$ which is the accuracy of M_i on a Full Test Set (Lomonaco & Maltoni, 2017). The Full Test Set is fixed and includes the patterns of all classes.

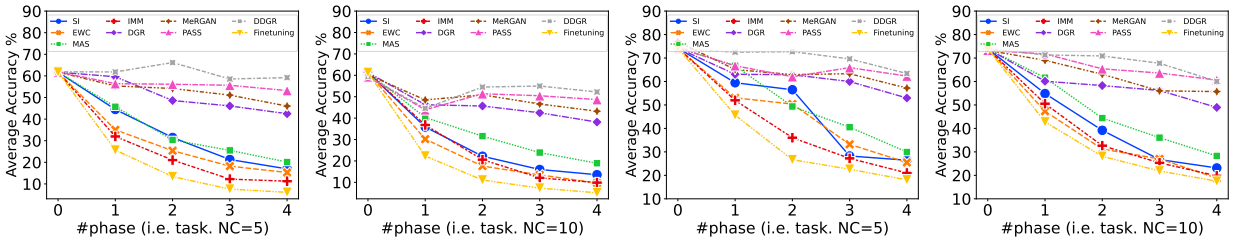
5.2. Results in CI scenario

We gather the final average accuracies and average forgetting rates of all experiments in the CI scenario in Table 1; for convenience, we refer to them as accuracy and forgetting, respectively. Moreover, to maintain consistency with (Liu et al., 2020), we refer to each task as a "phase" in the CI scenario. On CIFAR-100, DDGR achieves the best performance in almost all experiments. For example, when NC is 5 and the model is AlexNet, we observe that DDGR reduces the forgetting from 24.08% to 23.00% compared with the best baseline, representing an improvement of 1.08%, while DDGR outperforms the best baseline by 5.99% in terms of accuracy. We also track the performance of each baseline on CIFAR-100 in Figures 4(a) to 4(d). As the figures clearly show, our proposed DDGR outperforms the other baselines. For example, when NC is 5 and the model is AlexNet, the accuracy of DDGR is 0.05% to 10.04% higher than that of the best baseline at each phase; here, the accuracy of DDGR at each phase ranges from 58.57% to 66.13%.

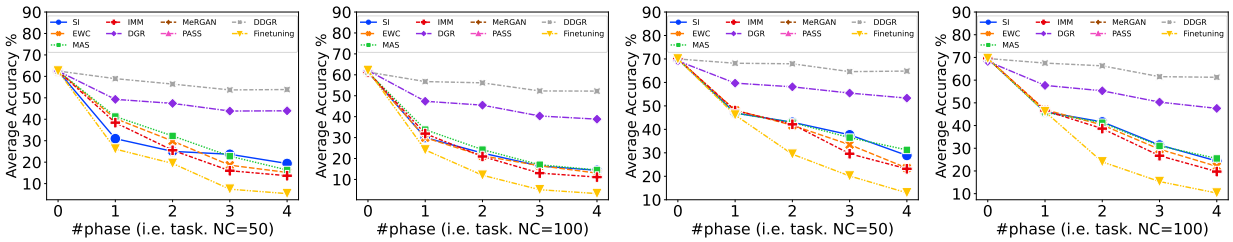
Table 1 further presents the results of all experiments on ImageNet. As is evident, DDGR performs better than all seven baselines. For example, when NC is 50 and the model is AlexNet, we find that DDGR improves the forgetting from 17.31% to 6.98% compared with the best baseline, representing an improvement of 10.33%. In the same experiment, DDGR outperforms the best baseline by 9.92% in terms of accuracy, achieving an overall accuracy of 53.86%. Figures 4(e) to 4(h) present the performance of all baselines on ImageNet. DDGR outperforms all baselines at every phase in all experiments. For example, when NC is 50 and the model is AlexNet, the accuracy of DDGR is 0.20% to 9.92% higher than that of the best baseline at each phase; here, the accuracy of DDGR at each phase ranges from 53.68% to 62.49%.

5.3. Results in CIR scenario

To maintain consistency with related works (Lomonaco & Maltoni, 2017), we refer to each task as a "batch" in this scenario. As illustrated in Figure 5, the performance of the model trained by each baseline is not stable, but the accuracy $acc_{i,FTS}$ tends to increase with new batches. This means that the model can accumulate the knowledge of

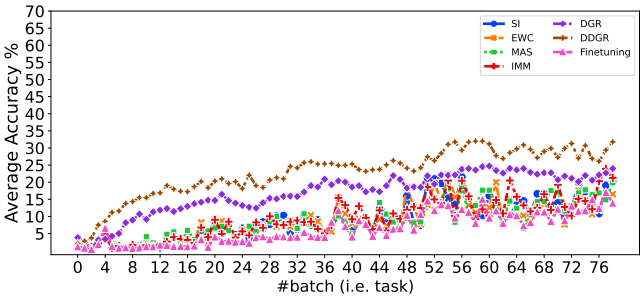


(a) CIFAR-100 AlexNet NC=5 (b) CIFAR-100 AlexNet NC=10 (c) CIFAR-100 ResNet NC=5 (d) CIFAR-100 ResNet NC=10

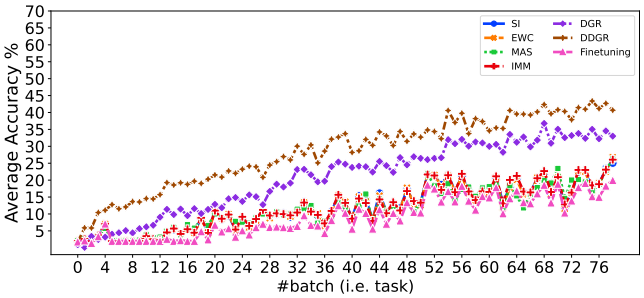


(e) ImageNet AlexNet NC=50 (f) ImageNet AlexNet NC=100 (g) ImageNet ResNet NC=50 (h) ImageNet ResNet NC=100

Figure 4. The curve of each baseline with different classifiers on CIFAR-100 and ImageNet. On CIFAR-100, there are 50 classes in the 0–th phase, and the number of classes (NC) of each subsequent phase is set to either 5 or 10. On ImageNet, there are 500 classes in the 0–th phase, and the number of classes (NC) of each subsequent phase is set to either 50 or 100.



(a) AlexNet



(b) ResNet

Figure 5. The results of using AlexNet and ResNet on CORe50 in CIR scenario. There are ten classes in the 0–th batch, and five classes in each subsequent phase.

classes in the CIR scenario. By checking the ending points of the curves in Figures 5(a) and 5(b), we can determine that the model trained by DDGR accumulates more knowledge about classes than those trained by other baselines. For example, the final accuracy $acc_{78,FTS}$ of DDGR using ResNet is 40.66%, which is 7.66% higher than that of the best baseline. Therefore, DDGR performs better than other baselines in the CIR scenario.

In addition, we demonstrate the advantages of DDGR in the experiments without a large first task (see more details in Appendix F). We present the effect of diffusion steps for DDGR in Appendix G. We also show that DDGR should be improved in terms of time cost in Appendix H.

6. Conclusion

In this paper, we propose a novel GR method called deep diffusion-based generative replay (DDGR). DDGR uses a diffusion model as the generator and proposes the *instruction-operator* to instruct the diffusion model, thereby generating high-quality samples for previous tasks. We demonstrate the advantages of DDGR by means of extensive experiments on CIFAR-100, ImageNet, and CORe50.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant 61976161, the Fundamental Research Funds for the Central Universities under Grant 2042022rc0016.

References

Achille, A., Eccles, T., Matthey, L., Burgess, C. P., Watters, N., Lerchner, A., and Higgins, I. Life-long disentangled representation learning with cross-domain latent homologies. In *NeurIPS*, pp. 9895–9905, 2018.

Ahn, H., Cha, S., Lee, D., and Moon, T. Uncertainty-based continual learning with adaptive regularization. In *NeurIPS*, pp. 4394–4404, 2019.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In *ECCV*, volume 11207, pp. 144–161, 2018.

Aljundi, R., Rohrbach, M., and Tuytelaars, T. Selfless sequential learning. In *ICLR*, 2019.

Ayub, A. and Wagner, A. R. EEC: learning to encode and regenerate images for continual learning. In *ICLR*, 2021.

Benjamin, A. S., Rolnick, D., and Körding, K. P. Measuring and regularizing networks in function space. In *ICLR*, 2019.

Cha, S., Hsu, H., Hwang, T., Calmon, F. P., and Moon, T. CPR: classifier-projection regularization for continual learning. In *ICLR*, 2021.

Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. Wavegrad: Estimating gradients for waveform generation. In *ICLR*, 2021.

Chen, Z. and Liu, B. *Lifelong Machine Learning, Second Edition*. 2018.

Choi, Y., El-Khamy, M., and Lee, J. Dual-teacher class-incremental learning with data-free generative replay. In *CVPR*, pp. 3543–3552, 2021.

Cossu, A., Graffieti, G., Pellegrini, L., Maltoni, D., Bacciu, D., Carta, A., and Lomonaco, V. Is class-incremental enough for continual learning? *Frontiers Artif. Intell.*, 5: 829842, 2022.

Croitoru, F., Hondru, V., Ionescu, R. T., and Shah, M. Diffusion models in vision: A survey. *CoRR*, abs/2209.04747, 2022.

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255, 2009.

Dhariwal, P. and Nichol, A. Q. Diffusion models beat gans on image synthesis. In *NeurIPS*, pp. 8780–8794, 2021.

Farquhar, S. and Gal, Y. Towards robust evaluations of continual learning. *CoRR*, abs/1805.09733, 2018.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. Generative adversarial nets. In *NeurIPS*, pp. 2672–2680, 2014.

Hayes, T. L., Krishnan, G. P., Bazhenov, M., Siegelmann, H. T., Sejnowski, T. J., and Kanan, C. Replay in deep learning: Current approaches and missing biological elements. *Neural Comput.*, 33(11):2908–2950, 2021.

Ho, J. and Salimans, T. Classifier-free diffusion guidance. *CoRR*, abs/2207.12598, 2022.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Ho, J., Saharia, C., Chan, W., Fleet, D. J., Norouzi, M., and Salimans, T. Cascaded diffusion models for high fidelity image generation. *J. Mach. Learn. Res.*, 23:47:1–47:33, 2022a.

Ho, J., Salimans, T., Gritsenko, A. A., Chan, W., Norouzi, M., and Fleet, D. J. Video diffusion models. *CoRR*, abs/2204.03458, 2022b.

Jing, B., Corso, G., Chang, J., Barzilay, R., and Jaakkola, T. S. Torsional diffusion for molecular conformer generation. *CoRR*, abs/2206.01729, 2022.

Jung, S., Ahn, H., Cha, S., and Moon, T. Continual learning with node-importance based adaptive group sparse regularization. In *NeurIPS*, 2020.

Kemker, R. and Kanan, C. Fearnet: Brain-inspired model for incremental learning. In *ICLR*, 2018.

Kim, H., Kim, S., and Yoon, S. Guided-tts: A diffusion model for text-to-speech via classifier guidance. In *ICML*, volume 162, pp. 11119–11133, 2022.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *ICLR*, 2014.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Kong, Z., Ping, W., Huang, J., Zhao, K., and Catanzaro, B. Diffwave: A versatile diffusion model for audio synthesis. In *ICLR*, 2021.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Kumaran, D., Hassabis, D., and McClelland, J. L. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

Lee, S., Kim, J., Jun, J., Ha, J., and Zhang, B. Overcoming catastrophic forgetting by incremental moment matching. In *NeurIPS*, pp. 4652–4662, 2017.

Lesort, T., Lomonaco, V., Stoian, A., Maltoni, D., Filliat, D., and Rodríguez, N. D. Continual learning for robotics. *CoRR*, abs/1907.00182, 2019.

Li, H., Yang, Y., Chang, M., Chen, S., Feng, H., Xu, Z., Li, Q., and Chen, Y. Srdiff: Single image super-resolution with diffusion probabilistic models. *Neurocomputing*, 479:47–59, 2022.

Liu, W. and Tsang, I. W. Large margin metric learning for multi-label prediction. In *AAAI*, pp. 2800–2806, 2015.

Liu, W. and Tsang, I. W. Making decision trees feasible in ultrahigh feature and label dimensions. *J. Mach. Learn. Res.*, 18:81:1–81:36, 2017.

Liu, W., Tsang, I. W., and Müller, K. An easy-to-hard learning paradigm for multiple classes and multiple labels. *J. Mach. Learn. Res.*, 18:94:1–94:38, 2017.

Liu, W., Xu, D., Tsang, I. W., and Zhang, W. Metric learning for multi-output tasks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(2):408–422, 2019.

Liu, Y., Su, Y., Liu, A., Schiele, B., and Sun, Q. Mnemonics training: Multi-class incremental learning without forgetting. In *CVPR*, pp. 12242–12251, 2020.

Lomonaco, V. and Maltoni, D. Core50: a new dataset and benchmark for continuous object recognition. In *CoRL*, 2017.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *NeurIPS*, pp. 6467–6476, 2017.

Luo, S. and Hu, W. Diffusion probabilistic models for 3d point cloud generation. In *CVPR*, pp. 2837–2845, 2021.

Luo, S., Shi, C., Xu, M., and Tang, J. Predicting molecular conformation via dynamic graph score matching. In *NeurIPS*, pp. 19784–19795, 2021.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, pp. 7765–7773, 2018.

Mundt, M., Pliushch, I., Majumder, S., Hong, Y. W., and Ramesh, V. Unified probabilistic deep continual learning through generative replay and open set recognition. *J. Imaging*, 8(4):93, 2022.

Nachmani, E., San-Roman, R., and Wolf, L. Non gaussian denoising diffusion models. *CoRR*, abs/2106.07582, 2021.

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *ICML*, volume 139, pp. 8162–8171, 2021.

Pan, P., Swaroop, S., Immer, A., Eschenhagen, R., Turner, R. E., and Khan, M. E. Continual deep learning by functional regularisation of memorable past. In *NeurIPS*, 2020.

Pandey, K., Mukherjee, A., Rai, P., and Kumar, A. Vaes meet diffusion models: Efficient and high-fidelity generation. In *NeurIPS*, 2021.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Pfülb, B. and Gepperth, A. A comprehensive, application-oriented study of catastrophic forgetting in dnns. In *ICLR*, 2019.

Popov, V., Vovk, I., Gogoryan, V., Sadekova, T., and Kudinov, M. A. Grad-tts: A diffusion probabilistic model for text-to-speech. In *ICML*, volume 139, pp. 8599–8608, 2021.

Ramapuram, J., Gregorova, M., and Kalousis, A. Lifelong generative modeling. *Neurocomputing*, 404:381–400, 2020.

Rebuffi, S., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *CVPR*, pp. 5533–5542, 2017.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. Experience replay for continual learning. In *NeurIPS*, pp. 348–358, 2019.

Saha, G., Garg, I., and Roy, K. Gradient projection memory for continual learning. In *ICLR*, 2021.

Serrà, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, volume 80, pp. 4555–4564, 2018.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In *NeurIPS*, pp. 2990–2999, 2017.

Singh, V., Jandial, S., Chopra, A., Ramesh, S., Krishnamurthy, B., and Balasubramanian, V. N. On conditioning the input noise for controlled image generation with diffusion models. *CoRR*, abs/2205.03859, 2022.

Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, volume 37, pp. 2256–2265, 2015.

Tang, B. and Matteson, D. S. Graph-based continual learning. In *ICLR*, 2021.

van de Ven, G. M. and Tolias, A. S. Three scenarios for continual learning. *CoRR*, abs/1904.07734, 2019.

van der Ven, M. and Tolias, A. S. Generative replay with feedback connections as a general strategy for continual learning. *CoRR*, abs/1809.10635, 2018.

Wu, C., Herranz, L., Liu, X., Wang, Y., van de Weijer, J., and Raducanu, B. Memory replay gans: learning to generate images from new categories without forgetting. *CoRR*, abs/1809.02058, 2018.

Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., and Fu, Y. Large scale incremental learning. In *CVPR*, pp. 374–382, 2019.

Xu, J. and Zhu, Z. Reinforced continual learning. In *NeurIPS*, pp. 907–916, 2018.

Xu, M., Yu, L., Song, Y., Shi, C., Ermon, S., and Tang, J. Geodiff: A geometric diffusion model for molecular conformation generation. In *ICLR*, 2022.

Ye, F. and Bors, A. G. Learning latent representations across multiple data domains using lifelong VAEGAN. In *ECCV*, volume 12365, pp. 777–795, 2020.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *ICML*, volume 70, pp. 3987–3995, 2017.

Zhai, M., Chen, L., Tung, F., He, J., Nawhal, M., and Mori, G. Lifelong GAN: continual learning for conditional image generation. In *ICCV*, pp. 2759–2768, 2019.

Zhu, F., Zhang, X., Wang, C., Yin, F., and Liu, C. Prototype augmentation and self-supervision for incremental learning. In *CVPR*, pp. 5871–5880, 2021.

A. Additional experimental setting

Here, we provide more information about baselines and datasets. There are 7 baselines in our paper. We give some descriptions of these baselines and introduce the datasets.

SI: The synaptic state tracks the past and current parameter value, and maintains an online estimate of the synapse's importance toward solving problems encountered in the past.

MAS: Redefines the parameter importance measure to an unsupervised setting and obtains gradients of the squared L_2-norm of the learned network output function.

EWC: EWC introduces network parameter uncertainty in the Bayesian framework; the true posterior is estimated using a Laplace approximation with precision, determined by the Fisher Information Matrix (FIM), which shows equivalence to the positive semi-definite second order derivative of the loss near a minimum.

IMM: IMM estimates Gaussian posteriors for task parameters, in the same vein as EWC, but inherently differs in its use of model merging. In the merging step, the mixture of Gaussian posteriors is approximated by a single Gaussian distribution and corresponding covariances.

DGR: DGR consists of a deep generative model ("generator") and a task solving model ("solver"). DGR trains a deep generative model in the generative adversarial networks (GANs) framework to mimic previous data. Generated data are then paired with corresponding response from the previous task solver to represent old tasks. Called the scholar model, the generator-solver pair can produce fake data and desired target pairs as much as needed, and when presented with a new task, these produced pairs are interleaved with new data to update the generator and solver networks.

MeRGAN: MeRGAN is a conditional GAN framework that integrates a memory replay generator. it contains two methods to prevent forgetting by leveraging replays, namely joint training with replay and replay alignment. In these paper, we use replay alignment.

PASS: PASS mainly consists of Prototype Augmentation and Self-Supervision. On the one hand, prototype augmentation (protoAug) memorizes one class representative prototype (typically the class mean in the deep feature space) for each old class, and augments the memorized prototypes via Gaussian noise when learning new classes. Then, the augmented prototypes and deep features of new data are jointly classified to maintain the discrimination and balance between old and new classes.

Finetuning: Finetuning greedily trains each task without considering previous task performance—hence introducing catastrophic forgetting—and represents the minimum desired performance.

Table A.1. Details of datasets

	Tasks	Classes/task	Train data/task	Task selection
CIFAR-100	5	$\{50, 5\}$ or $\{50, 10\}$	$\{25000, 2500\}$ or $\{25000, 5000\}$	random class
ImageNet	5	$\{500, 50\}$ or $\{500, 100\}$	$\{650000, 65000\}$ or $\{650000, 130000\}$	random class
CORe50	79	$\{10, 5\}$	$\{3000, 1500\}$	random class

Datasets: There are three datasets in our work, we divide them into several tasks which can be found in Table A.1. In CI, we divide the datasets into 5 tasks where there are 1 initial task and 4 incremental tasks. For CIFAR-100, the initial task consist of 50 random classes, and all incremental tasks have same numbers of classes. The number of classes in one incremental task is set to be 5 or 10. For ImageNet, the initial task consist of 500 random classes, and the number of classes in one incremental task is set to be 50 or 100. In CIR, CORe50 is a collection of 50 domestic objects belonging to 10 categories. Classification in this paper is performed at object level (50 classes). We follow up the work of (Lomonaco & Maltoni, 2017) which generates 79 tasks from CORe50. The first task includes 10 classes, and each subsequent task contains 5 classes.

B. Reparameterization trick

In this section, we provide details of reparameterization trick in forward process of DDPM. A nice property of the forward process is that we can sample $\mathbf{x}_n$ at any time step n in a closed form using reparameterization trick:

$$\mathbf{x}_n = \sqrt{\alpha_n}\mathbf{x}_{n-1} + \sqrt{1-\alpha_n}\boldsymbol{\epsilon}_{n-1} \tag{B.1}$$

$$\mathbf{x}_{n-1} = \sqrt{\alpha_{n-1}}\mathbf{x}_{n-2} + \sqrt{1-\alpha_{n-1}}\boldsymbol{\epsilon}_{n-2} \tag{B.2}$$

where $\alpha_n = 1-\beta_n$ and $\boldsymbol{\epsilon}_{n-1}, \boldsymbol{\epsilon}_{n-2} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Pluging (B.2) into (B.1), we derive that:

$$\mathbf{x}_n = \sqrt{\alpha_n\alpha_{n-1}}\mathbf{x}_{n-2} + \sqrt{\alpha_n(1-\alpha_{n-1})}\boldsymbol{\epsilon}_{n-2} + \sqrt{1-\alpha_n}\boldsymbol{\epsilon}_{n-1} \tag{B.3}$$

where $\sqrt{\alpha_n(1-\alpha_{n-1})}\boldsymbol{\epsilon}_{n-2}$ corresponds to $\mathcal{N}\left(\mathbf{0}, \alpha_n(1-\alpha_{n-1})\mathbf{I}\right)$ and $\sqrt{1-\alpha_n}\boldsymbol{\epsilon}_{n-1}$ corresponds to $\mathcal{N}\left(\mathbf{0}, 1-\alpha_n\mathbf{I}\right)$. By merging these two distributions, we have the new distribution $\mathcal{N}\left(\mathbf{0}, 1-\alpha_n\alpha_{n-1}\mathbf{I}\right)$ with merged standard deviation $\sqrt{\alpha_n(1-\alpha_{n-1}) + (1-\alpha_n)} = \sqrt{1-\alpha_n\alpha_{n-1}}$. (B.3) can be rewritten as:

$$\mathbf{x}_n = \sqrt{\alpha_n\alpha_{n-1}}\mathbf{x}_{n-2} + \sqrt{1-\alpha_n\alpha_{n-1}}\overline{\boldsymbol{\epsilon}}_{n-2} \tag{B.4}$$

where $\sqrt{1-\alpha_n\alpha_{n-1}}\overline{\boldsymbol{\epsilon}}_{n-2}$ merges two Gaussians $\mathcal{N}\left(\mathbf{0}, \alpha_n(1-\alpha_{n-1})\mathbf{I}\right)$ and $\mathcal{N}\left(\mathbf{0}, 1-\alpha_n\mathbf{I}\right)$. After n iterations, we can use $\mathbf{x}_0$ to represent $\mathbf{x}_n$:

$$\mathbf{x}_n = \sqrt{\bar{\alpha}_n}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_n}\boldsymbol{\epsilon} \tag{B.5}$$

where $\bar{\alpha}_n = \prod_{i=1}^{n}\alpha_i$. These also indicate that:

$$q\left(\mathbf{x}_n \mid \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_n; \sqrt{\bar{\alpha}_n}\mathbf{x}_0, \left(1-\bar{\alpha}_n\right)\mathbf{I}\right) \tag{B.6}$$

C. Additional derivation of training objective

There are two ways to derive the lower bound L_{LB}. First, as shown in the main file, we can leverage the variational lower bound to optimize the negative log-likelihood:

$$\begin{aligned}
-\log p_\theta\left(\mathbf{x}_0\right) &\leq -\log p_\theta\left(\mathbf{x}_0\right) + D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)\right) && \text{(C.7)}\\
&= -\log p_\theta\left(\mathbf{x}_0\right) + \mathbb{E}_{\mathbf{x}_{1:N}\sim q(\mathbf{x}_{1:N}|\mathbf{x}_0)}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)/p_\theta\left(\mathbf{x}_0\right)}\right] && \text{(C.8)}\\
&= -\log p_\theta\left(\mathbf{x}_0\right) + \mathbb{E}_{q(\mathbf{x}_{1:N}|\mathbf{x}_0)}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)} + \log p_\theta\left(\mathbf{x}_0\right)\right] && \text{(C.9)}\\
&= \mathbb{E}_{q(\mathbf{x}_{1:N}|\mathbf{x}_0)}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)}\right] && \text{(C.10)}
\end{aligned}$$

$$-\mathbb{E}_{q(\mathbf{x}_0)}\log p_\theta\left(\mathbf{x}_0\right) \leq \mathbb{E}_{q(\mathbf{x}_{0:N})}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)}\right] =: L_{LB} \tag{C.11}$$

The other way to derive L_{LB} is to minimize the cross entropy as the training objective:

$$\begin{aligned}
L_{\mathrm{CE}} &= -\mathbb{E}_{q(\mathbf{x}_0)}\log p_\theta\left(\mathbf{x}_0\right) && \text{(C.12)}\\
&= -\mathbb{E}_{q(\mathbf{x}_0)}\log\left(\int p_\theta\left(\mathbf{x}_{0:N}\right)d\mathbf{x}_{1:N}\right) && \text{(C.13)}\\
&= -\mathbb{E}_{q(\mathbf{x}_0)}\log\left(\int q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)\frac{p_\theta\left(\mathbf{x}_{0:N}\right)}{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}d\mathbf{x}_{1:N}\right) && \text{(C.14)}\\
&= -\mathbb{E}_{q(\mathbf{x}_0)}\log\left(\mathbb{E}_{q(\mathbf{x}_{1:N}|\mathbf{x}_0)}\frac{p_\theta\left(\mathbf{x}_{0:N}\right)}{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}\right) && \text{(C.15)}\\
&\leq -\mathbb{E}_{q(\mathbf{x}_{0:N})}\log\frac{p_\theta\left(\mathbf{x}_{0:N}\right)}{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)} && \text{(C.16)}\\
&= \mathbb{E}_{q(\mathbf{x}_{0:N})}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)}\right] = L_{\mathrm{LB}} && \text{(C.17)}
\end{aligned}$$

where (C.16) uses the Jensen's inequality directly. Sohl-Dickstein et al. (2015) split the objective L_{LB} into several KL-divergence terms where each term in the objective is analytically computable:

$$
\begin{aligned}
L_{LB} &= \mathbb{E}_{q(\mathbf{x}_{0:N})}\left[\log \frac{q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:N}\right)}\right] && \text{(C.18)}\\
&= \mathbb{E}_q\left[\log \frac{\prod_{n=0}^{N-1} q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right)}{p_\theta\left(\mathbf{x}_N\right) \prod_{n=0}^{N-1} p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)}\right] && \text{(C.19)}\\
&= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_N\right)+\sum_{n=0}^{N-1} \log \frac{q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right)}{p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)}\right] && \text{(C.20)}\\
&= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_N\right)+\sum_{n \geq 1} \log \frac{q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right)}{p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)}+\log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}\right] && \text{(C.21)}\\
&= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_N\right)+\sum_{n \geq 1} \log \left(\frac{q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)} \cdot \frac{q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_n \mid \mathbf{x}_0\right)}\right)+\log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}\right] && \text{(C.22)}\\
&= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_N\right)+\sum_{n \geq 1} \log \frac{q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)}+\sum_{n \geq 1} \log \frac{q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_n \mid \mathbf{x}_0\right)}+\log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}\right] && \text{(C.23)}\\
&= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_N\right)+\sum_{n \geq 1} \log \frac{q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)}+\log \frac{q\left(\mathbf{x}_N \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}+\log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}\right] && \text{(C.24)}\\
&= \mathbb{E}_q\left[\log \frac{q\left(\mathbf{x}_N \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_N\right)}+\sum_{n \geq 1} \log \frac{q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)}-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)\right] && \text{(C.25)}\\
&= \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_N \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_N\right)\right)}_{L_N}+\sum_{n \geq 1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)\right)}_{L_n} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]. && \text{(C.26)}
\end{aligned}
$$

DDPM fixes β_n to constants. L_N can be ignored because L_N is also constant—no learnable parameters. In the work of Ho et al. (2020), L_0 is modelled by a separate discrete decoder derived from $\mathcal{N}\left(\mathbf{x}_0; \boldsymbol{\mu}_\theta\left(\mathbf{x}_1, 1\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_1, 1\right)\right)$. As for L_n ($n=1, \cdots, N-1$), it is worth noting that $q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right)$ of L_n is traceable:

$$
q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right)=\mathcal{N}\left(\mathbf{x}_n ; \tilde{\boldsymbol{\mu}}_{n+1}, \tilde{\boldsymbol{\Sigma}}_{n+1} \mathbf{I}\right) \tag{C.27}
$$

According to Bayes' rule, we can derive that:

$$
\begin{aligned}
& q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, \mathbf{x}_0\right) && \text{(C.28)}\\
&= q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n, \mathbf{x}_0\right) \frac{q\left(\mathbf{x}_n \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_0\right)} && \text{(C.29)}\\
&\propto \exp\left(-\frac{1}{2}\left(\frac{\left(\mathbf{x}_{n+1}-\sqrt{\alpha_{n+1}} \mathbf{x}_n\right)^2}{\beta_{n+1}}+\frac{\left(\mathbf{x}_n-\sqrt{\bar{\alpha}_n} \mathbf{x}_0\right)^2}{1-\bar{\alpha}_n}-\frac{\left(\mathbf{x}_{n+1}-\sqrt{\bar{\alpha}_{n+1}} \mathbf{x}_0\right)^2}{1-\bar{\alpha}_{n+1}}\right)\right) && \text{(C.30)}\\
&= \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}_{n+1}^2-2 \sqrt{\alpha_{n+1}} \mathbf{x}_{n+1} \mathbf{x}_n+\alpha_{n+1} \mathbf{x}_n^2}{\beta_{n+1}}+\frac{\mathbf{x}_n^2-2 \sqrt{\bar{\alpha}_n} \mathbf{x}_0 \mathbf{x}_n+\bar{\alpha}_n \mathbf{x}_0^2}{1-\bar{\alpha}_n}-\frac{\left(\mathbf{x}_{n+1}-\sqrt{\bar{\alpha}_{n+1}} \mathbf{x}_0\right)^2}{1-\bar{\alpha}_{n+1}}\right)\right) && \text{(C.31)}\\
&= \exp\left(-\frac{1}{2}\left(\underbrace{\left(\frac{\alpha_{n+1}}{\beta_{n+1}}+\frac{1}{1-\bar{\alpha}_n}\right)}_{1 / \tilde{\boldsymbol{\Sigma}}_{n+1}} \mathbf{x}_n^2-\underbrace{\left(\frac{2 \sqrt{\alpha_{n+1}}}{\beta_{n+1}} \mathbf{x}_{n+1}+\frac{2 \sqrt{\bar{\alpha}_n}}{1-\bar{\alpha}_n} \mathbf{x}_0\right)}_{2 \tilde{\boldsymbol{\mu}}_{n+1} / \tilde{\boldsymbol{\Sigma}}_{n+1}} \mathbf{x}_n+C\left(\mathbf{x}_{n+1}, \mathbf{x}_0\right)\right)\right) && \text{(C.32)}
\end{aligned}
$$

where $C\left(\mathbf{x}_{n+1}, \mathbf{x}_0\right)$ is not involved with $\mathbf{x}_n$ and can be ignored. According to the standard Gaussian density function, we can achieve the mean:

$$\tilde{\boldsymbol{\mu}}_{n+1} = \left(\frac{\sqrt{\alpha_{n+1}}}{\beta_{n+1}}\mathbf{x}_{n+1} + \frac{\sqrt{\bar{\alpha}_n}}{1-\bar{\alpha}_n}\mathbf{x}_0\right) / \left(\frac{\alpha_{n+1}}{\beta_{n+1}} + \frac{1}{1-\bar{\alpha}_n}\right) \tag{C.33}$$

$$= \left(\frac{\sqrt{\alpha_{n+1}}}{\beta_{n+1}}\mathbf{x}_{n+1} + \frac{\sqrt{\bar{\alpha}_n}}{1-\bar{\alpha}_n}\mathbf{x}_0\right) \frac{1-\bar{\alpha}_n}{1-\bar{\alpha}_{n+1}} \cdot \beta_{n+1} \tag{C.34}$$

$$= \frac{\sqrt{\alpha_{n+1}}\left(1-\bar{\alpha}_n\right)}{1-\bar{\alpha}_{n+1}}\mathbf{x}_{n+1} + \frac{\sqrt{\bar{\alpha}_n}\beta_{n+1}}{1-\bar{\alpha}_{n+1}}\mathbf{x}_0 \tag{C.35}$$

Recall that $\mathbf{x}_{n+1} = \sqrt{\bar{\alpha}_{n+1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{n+1}}\boldsymbol{\epsilon}$, we have $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_{n+1}}}\left(\mathbf{x}_{n+1} - \sqrt{1-\bar{\alpha}_{n+1}}\boldsymbol{\epsilon}\right)$. Pluging it in (C.35), we derive that:

$$\tilde{\boldsymbol{\mu}}_{n+1} = \frac{\sqrt{\alpha_{n+1}}\left(1-\bar{\alpha}_n\right)}{1-\bar{\alpha}_{n+1}}\mathbf{x}_{n+1} + \frac{\sqrt{\bar{\alpha}_n}\beta_{n+1}}{1-\bar{\alpha}_{n+1}}\frac{1}{\sqrt{\bar{\alpha}_{n+1}}}\left(\mathbf{x}_{n+1} - \sqrt{1-\bar{\alpha}_{n+1}}\boldsymbol{\epsilon}\right) \tag{C.36}$$

$$= \frac{1}{\sqrt{\alpha_{n+1}}}\left(\mathbf{x}_{n+1} - \frac{1-\alpha_{n+1}}{\sqrt{1-\bar{\alpha}_{n+1}}}\boldsymbol{\epsilon}\right) \tag{C.37}$$

The variance can also be achieved:

$$\tilde{\boldsymbol{\Sigma}}_{n+1} = 1/\left(\frac{\alpha_{n+1}}{\beta_{n+1}} + \frac{1}{1-\bar{\alpha}_n}\right) = \frac{1-\bar{\alpha}_n}{1-\bar{\alpha}_{n+1}} \cdot \beta_{n+1} \tag{C.38}$$

In DDPM, $\tilde{\boldsymbol{\Sigma}}_{n+1}$ is set to $\sigma_{n+1}^2\mathbf{I}$ which are untrained time dependent constants. These mean that it only needs to train $\boldsymbol{\mu}_\theta\left(\mathbf{x}_{n+1}, n+1\right)$ to predict $\tilde{\boldsymbol{\mu}}_{n+1}$. As $\mathbf{x}_{n+1}$ can be achieved at training time, it only needs to train $\boldsymbol{\epsilon}_\theta(\mathbf{x}_{n+1}, n+1)$ to predict $\boldsymbol{\epsilon}$. By using a specific parameterization, we achieve:

$$L_n \tag{C.39}$$

$$= \mathbb{E}_q\left[\frac{1}{2\sigma_{n+1}^2}\left\|\tilde{\boldsymbol{\mu}}_{n+1} - \boldsymbol{\mu}_\theta\left(\mathbf{x}_{n+1}, n+1\right)\right\|^2\right] + C \tag{C.40}$$

$$= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{1}{2\sigma_{n+1}^2}\left\|\frac{1}{\sqrt{\alpha_{n+1}}}\left(\mathbf{x}_{n+1} - \frac{1-\alpha_{n+1}}{\sqrt{1-\bar{\alpha}_{n+1}}}\boldsymbol{\epsilon}\right) - \frac{1}{\sqrt{\alpha_{n+1}}}\left(\mathbf{x}_{n+1} - \frac{1-\alpha_{n+1}}{\sqrt{1-\bar{\alpha}_{n+1}}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_{n+1}, n+1)\right)\right\|^2\right] + C \tag{C.41}$$

$$= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{\left(1-\alpha_{n+1}\right)^2}{2\sigma_{n+1}^2\alpha_{n+1}\left(1-\bar{\alpha}_{n+1}\right)}\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_{n+1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{n+1}}\boldsymbol{\epsilon}, n+1\right)\right\|^2\right] + C \tag{C.42}$$

where C is a constant that does not depend on θ. Ho et al. (2020) find that it is better to train the diffusion model without the weighting term. The final objective is:

$$\mathbb{E}_{n\sim[1,N-1],\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_{n+1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{n+1}}\boldsymbol{\epsilon}, n+1\right)\right\|^2\right] \tag{C.43}$$

D. Proof of proportionality

Here we provide the proof of Theorem 4.1 that $\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y\right)$ is proportional to $q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right)\hat{q}\left(y \mid \mathbf{x}_n\right)$. We first define the Markov process $\hat{q}$ which is conditioned on label y:

$$\begin{aligned}
&\hat{q}\left(\mathbf{x}_0\right) := q\left(\mathbf{x}_0\right), \quad \hat{q}\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0, y\right) := \prod_{n=1}^{N}\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n-1}, y\right), \\
&\hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n, y\right) := \mathcal{N}\left(\mathbf{x}_n; \sqrt{1-\beta_n}\mathbf{x}_{n-1}, \beta_n\mathbf{I}\right) = q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right), \\
&\hat{q}\left(y \mid \mathbf{x}_0\right) := \text{a label distribution on } \mathbf{x}_0 \text{ which is known.}
\end{aligned} \tag{D.44}$$

Next, we prove that the conditional process $\hat{q}$ has the following properties.

Lemma D.1. *The conditional process $\hat{q}$ has the following properties:*

$$\hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) = q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) \tag{D.45}$$
$$\hat{q}\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right) = q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right) \tag{D.46}$$
$$\hat{q}\left(\mathbf{x}_n\right) = q\left(\mathbf{x}_n\right) \tag{D.47}$$
$$\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) = q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) \tag{D.48}$$
$$\hat{q}\left(y \mid \mathbf{x}_n, \mathbf{x}_{n+1}\right) = \hat{q}\left(y \mid \mathbf{x}_n\right) \tag{D.49}$$

Proof. We first prove Equation (D.45):

$$\hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) = \int_y \hat{q}\left(\mathbf{x}_{n+1}, y \mid \mathbf{x}_n\right) dy \tag{D.50}$$
$$= \int_y \hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n, y\right) \hat{q}\left(y \mid \mathbf{x}_n\right) dy \tag{D.51}$$
$$= \int_y q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) \hat{q}\left(y \mid \mathbf{x}_n\right) dy \tag{D.52}$$
$$= q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) \int_y \hat{q}\left(y \mid \mathbf{x}_n\right) dy \tag{D.53}$$
$$= q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) \tag{D.54}$$

Following the similar proof with (D.45), we can prove Equation (D.46):

$$\hat{q}\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right) = \int_y \hat{q}\left(\mathbf{x}_{1:N}, y \mid \mathbf{x}_0\right) dy \tag{D.55}$$
$$= \int_y \hat{q}\left(y \mid \mathbf{x}_0\right) \hat{q}\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0, y\right) dy \tag{D.56}$$
$$= \int_y \hat{q}\left(y \mid \mathbf{x}_0\right) \prod_{n=1}^{N} \hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n-1}, y\right) dy \tag{D.57}$$
$$= \int_y \hat{q}\left(y \mid \mathbf{x}_0\right) \prod_{n=1}^{N} q\left(\mathbf{x}_n \mid \mathbf{x}_{n-1}\right) dy \tag{D.58}$$
$$= \prod_{n=1}^{N} q\left(\mathbf{x}_n \mid \mathbf{x}_{n-1}\right) \int_y \hat{q}\left(y \mid \mathbf{x}_0\right) dy \tag{D.59}$$
$$= \prod_{n=1}^{N} q\left(\mathbf{x}_n \mid \mathbf{x}_{n-1}\right) \tag{D.60}$$
$$= q\left(\mathbf{x}_{1:N} \mid \mathbf{x}_0\right) \tag{D.61}$$

where Equation (D.57) uses the definition in (D.44) to derive Equation (D.58). By using Equation (D.46), we can prove Equation (D.47):

$$\hat{q}\left(\mathbf{x}_n\right) = \int_{\mathbf{x}_{0:n-1}} \hat{q}\left(\mathbf{x}_0, \ldots, \mathbf{x}_n\right) d\mathbf{x}_{0:n-1} \tag{D.62}$$
$$= \int_{\mathbf{x}_{0:n-1}} \hat{q}\left(\mathbf{x}_0\right) \hat{q}\left(\mathbf{x}_1, \ldots, \mathbf{x}_n \mid \mathbf{x}_0\right) d\mathbf{x}_{0:n-1} \tag{D.63}$$
$$= \int_{\mathbf{x}_{0:n-1}} q\left(\mathbf{x}_0\right) q\left(\mathbf{x}_1, \ldots, \mathbf{x}_n \mid \mathbf{x}_0\right) d\mathbf{x}_{0:n-1} \tag{D.64}$$
$$= \int_{\mathbf{x}_{0:n-1}} q\left(\mathbf{x}_0, \ldots, \mathbf{x}_n\right) d\mathbf{x}_{0:n-1} \tag{D.65}$$
$$= q\left(\mathbf{x}_n\right) \tag{D.66}$$

Combining (D.45) with (D.47), we can directly use Bayes' rule to prove Equation (D.48):

$$\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) = \hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) \hat{q}\left(\mathbf{x}_n\right) / \hat{q}\left(\mathbf{x}_{n+1}\right) \tag{D.67}$$
$$= q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) q\left(\mathbf{x}_n\right) / q\left(\mathbf{x}_{n+1}\right) \tag{D.68}$$
$$= q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) \tag{D.69}$$

By using the definition $\hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n, y\right) = q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right)$ in (D.44) and Equation (D.48), we can prove Equation (D.49):

$$\hat{q}\left(y \mid \mathbf{x}_n, \mathbf{x}_{n+1}\right) = \hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n, y\right) \frac{\hat{q}\left(y \mid \mathbf{x}_n\right)}{\hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right)} \tag{D.70}$$
$$= q\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) \frac{\hat{q}\left(y \mid \mathbf{x}_n\right)}{\hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right)} \tag{D.71}$$
$$= \hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right) \frac{\hat{q}\left(y \mid \mathbf{x}_n\right)}{\hat{q}\left(\mathbf{x}_{n+1} \mid \mathbf{x}_n\right)} \tag{D.72}$$
$$= \hat{q}\left(y \mid \mathbf{x}_n\right) \tag{D.73}$$

□

Finally, by combining (D.48) and (D.49) in Lemma D.1, we can track the conditional probability $\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y\right)$:

$$\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y\right) = \frac{\hat{q}\left(\mathbf{x}_n, \mathbf{x}_{n+1}, y\right)}{\hat{q}\left(\mathbf{x}_{n+1}, y\right)} \tag{D.74}$$
$$= \frac{\hat{q}\left(\mathbf{x}_n, \mathbf{x}_{n+1}, y\right)}{\hat{q}\left(y \mid \mathbf{x}_{n+1}\right) \hat{q}\left(\mathbf{x}_{n+1}\right)} \tag{D.75}$$
$$= \frac{\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) \hat{q}\left(y \mid \mathbf{x}_n, \mathbf{x}_{n+1}\right) \hat{q}\left(\mathbf{x}_{n+1}\right)}{\hat{q}\left(y \mid \mathbf{x}_{n+1}\right) \hat{q}\left(\mathbf{x}_{n+1}\right)} \tag{D.76}$$
$$= \frac{\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) \hat{q}\left(y \mid \mathbf{x}_n, \mathbf{x}_{n+1}\right)}{\hat{q}\left(y \mid \mathbf{x}_{n+1}\right)} \tag{D.77}$$
$$= \frac{\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) \hat{q}\left(y \mid \mathbf{x}_n\right)}{\hat{q}\left(y \mid \mathbf{x}_{n+1}\right)} \tag{D.78}$$
$$= \frac{q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) \hat{q}\left(y \mid \mathbf{x}_n\right)}{\hat{q}\left(y \mid \mathbf{x}_{n+1}\right)} \tag{D.79}$$

As $\hat{q}\left(y \mid \mathbf{x}_{n+1}\right)$ is independent from $\mathbf{x}_n$, it can be regarded as a constant. Therefore, $\hat{q}\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}, y\right)$ is proportional to $q\left(\mathbf{x}_n \mid \mathbf{x}_{n+1}\right) \hat{q}\left(y \mid \mathbf{x}_n\right)$. Dhariwal & Nichol (2021) also provide a similar proof which can also be referred to.

E. Samples generated by DDGR

Here we present the samples generated by DDGR at the last three batches. As illustrated in Figure E.1, the samples generated by DGR are still clear.

Figure E.1. The samples generated by DDGR at the last three batches.

F. Experiments without a large first task

To further demonstrate the advantages of DDGR, we try to conduct experiments on CIFAR100 without a large first task by AlexNet. We split CIFAR100 into five tasks, and each task has 20 classes. The chosen baseline is PASS which outputs other baselines in Section 5.2. The results are presented in Table F.2.

Table F.2. The average accuracy A_i of each phase on CIFAR100 where all tasks have the same number (20) of classes.

	A_i				
	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4
PASS	70.51	52.17	50.41	47.39	44.32
DDGR	70.65	54.33	53.35	51.16	49.03

As shown in Table F.2, DDGR also achieves the best performance in all experiments. For example, DDGR outperforms the best baseline by 4.71% in terms of the final average accuracy. The average accuracy of DDGR is 0.14% to 4.71% higher than that of the best baseline at each phase. Therefore, DDGR still works without a large first task.

G. The effect of diffusion steps

To figure out the effect of diffusion steps, we conduct experiments and present the results with numbers of different diffusion steps in Table G.3. Overall, the gap of the final average accuracy and forgetting rate between different N is small. For example, when NC (number of classes in the incremental tasks) is 5, the final average accuracy ranges from 56.66% to 59.20%, representing a 2.54% difference.

Table G.3. We present the final average accuracy A_T and forgetting rate F_T^{avg} on CIFAR100 with different numbers of diffusion model steps. The classifier is AlexNet.

	A_T		F_T^{avg}	
N	NC=5	10	5	10
1000	56.66	50.20	24.83	19.43
2000	59.02	52.78	22.78	15.10
3000	56.80	51.10	24.30	19.52
4000	59.20	52.22	23.00	16.86

H. Time cost of DDGR

We present the time cost in Table H.4, which indicates that DDGR should be improved in terms of the time cost.

Table H.4. The total time cost (s) of baselines on CIFAR100.

	Time cost (s)	
method	NC=5	10
Finetuning	2421.40	3541.63
SI	3638.51	5167.41
EWC	5229.43	7114.12
MAS	6923.60	8491.71
IMM	8054.56	10324.09
DGR	69532.44	72543.57
DDGR	75692.41	87514.23

We count the time cost of all experiments on CIFAR100. Generally speaking, DDGR spends more time than other baselines. In Table H.5, we further analyse the time cost of DDGR, and find that the training and sampling process of diffusion model cost too much time while the calculation of instruction-operator only spends a little time. These indicate that DDGR spends a lot of time due to the training and sampling process of diffusion model rather than the calculation of our proposed instruction-operator.

Table H.5. The every part of time cost (s) of DDGR on CIFAR100.

	Time cost (s)									
	NC=5					NC=10				
	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4
Instruction-operator (IO)	0.00	872.13	901.54	1024.11	957.55	0.00	949.41	834.19	1064.48	950.40
Sampling (except IO)	0.00	8589.24	8891.10	10539.16	9433.80	0.00	9448.09	8658.69	10937.98	9679.87
Classifier training	223.93	109.23	104.16	155.62	139.16	1029.84	478.96	760.62	752.82	1149.98
Diffusion model training	12776.03	4943.12	4033.18	6130.99	5868.34	13396.11	5903.28	6940.05	7519.57	7059.89
Sum	12999.96	14513.72	13929.98	17849.88	16398.86	14425.95	16779.74	17193.55	20274.85	18840.14