BooVAE: Boosting Approach for Continual Learning of VAE

Evgenii Egorov *†
University of Amsterdam
egorov.evgenyy@ya.ru

Anna Kuzina *†
Vrije Universiteit
av.kuzina@yandex.ru

Evgeny Burnaev
Skoltech, AIRI
e.burnaev@skoltech.ru

Abstract

Variational autoencoder (VAE) is a deep generative model for unsupervised learning, allowing to encode observations into the meaningful latent space. VAE is prone to catastrophic forgetting when tasks arrive sequentially, and only the data for the current one is available. We address this problem of continual learning for VAEs. It is known that the choice of the prior distribution over the latent space is crucial for VAE in the non-continual setting. We argue that it can also be helpful to avoid catastrophic forgetting. We learn the approximation of the aggregated posterior as a prior for each task. This approximation is parametrised as an additive mixture of distributions induced by encoder evaluated at trainable pseudo-inputs. We use a greedy boosting-like approach with entropy regularisation to learn the components. This method encourages components diversity, which is essential as we aim at memorising the current task with the fewest components possible. Based on the learnable prior, we introduce an end-to-end approach for continual learning of VAEs and provide empirical studies on commonly used benchmarks (MNIST, Fashion MNIST, NotMNIST) and CelebA datasets. For each dataset, the proposed method avoids catastrophic forgetting in a fully automatic way.

1 Introduction

Variational Autoencoders (VAEs) (Kingma & Welling, 2013) are deep generative models used in various domains (Lee et al., 2017; Zhou et al., 2020). The VAE model consists of deep neural networks (DNNs): an *encoder* (inference network) and *decoder* (generative network). DNNs are known to reduce their quality on previously learned tasks when trained on data from a new task. Several directions to address this problem of *catastrophic forgetting* were suggested. But this phenomenon is mainly considered without attention to the specific properties of the VAE. We want to discuss current approaches to continual learning and formulate requirements for the ideal solution.

Dynamic architecture approach adds task-specific last layers (multi-heads) to encoder and decoder for each task (Rusu et al., 2016; Nguyen et al., 2018; Li & Hoiem, 2018). We suppose that practical applications of VAE require the common latent space, which is violated in multi-heads. Using multi-heads requires deciding which head to apply to the new data and when to expand the architecture. They reduce reuse of similarities between tasks. Hence, we suppose that the approach *should keep the static architecture* for both encoder and decoder.

The generative replay (Shin et al., 2017; Rao et al., 2019) uses a "teacher" generative model for generating "fake" data that mimics former training examples. Then the "student" model is trained on joint "fake" and new data. This approach is conceptually simple, model-agnostic and overcomes forgetting. However, these benefits come with a computational price. We need to retrain model while generating the dataset from all the past tasks, asses samples quality and the task-balance. Thus the approach *should avoid generative replay*).

*These authors contributed equally

†The work was done while the author was at the Skoltech, Moscow, 2019

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Weight penalty approach (Liu et al., 2018; Kirkpatrick et al., 2017) forms the trust-region around the optimum of the previous task for protecting parameters. This approach preserves the architecture and avoids generative replay. However, for DNNs a change in the weights is a poor proxy for the difference in the outputs (Benjamin et al., 2018). It is an even more critical issue for VAE model as it consists of a pair DNNs: encoder and decoder. Thus the approach *should link the data-space and the latent-space*.

We propose a novel continual learning approach for the VAEs. For each task, we expand current prior to get the approximation of an aggregated posterior over the whole data. We parametrise approximation as an additive mixture of distributions induced by encoder evaluated at trainable pseudo-inputs. These pseudo-inputs link the data-space and latent-space and help to memorise knowledge about past tasks. The problem of matching the aggregated posterior is ill-posed since we observe only its empirical version. As a solution, we use a greedy boosting-like approach with entropy regularisation. This method encourages components in the learned approximation to be diverse, which is essential as we aim at memorising the current task with the fewest components possible. The proposed approach is orthogonal to other methods mentioned above and can be applied in combination with them. Our main contributions are the following:

- We relate the approximation of optimal prior, the aggregated posterior and the continual learning task for VAE model. We find optimal additive perturbation in order to approximate optimal prior distribution. We derive the algorithm of effective approximation of the optimal prior for the continual learning framework.
- We use this result and present *Boosting Approach for Continual Learning of VAE* (BooVAE), a framework for training VAE models in the continual framework with static architecture.
- We empirically validate the proposed algorithm on commonly used benchmarks (MNIST, Fashion-MNIST, NotMNIST) and CelebA for disjoint sequential image generation tasks. The proposed generative model could be efficiently used in a generative replay for discriminative models. We train both generative and discriminative models incrementally, avoiding retraining the generative model for each task from scratch or storing several generative models. We provide code at `https://github.com/AKuzina/BooVAE`.

2 Background

Variational Autoencoders (VAEs) Let $p_\theta(\boldsymbol{x}, \boldsymbol{z})$ be the joint distribution of observed variable $\boldsymbol{x} \in \mathbb{R}^D$ and hidden latent variable $\boldsymbol{z}$, with the distribution $\pi(\boldsymbol{z})$. Given the distribution $\boldsymbol{x} \sim p_e(\boldsymbol{x})$ we aim to find θ which maximizes the marginal log-likelihood $\mathbb{E}_{p_e(\boldsymbol{x})}\left[\log \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}\right]$. Since the marginal likelihood is often intractable, we solve this optimization problem by the variational inference (Jordan et al., 1999). Variational autoencoders (VAE) (Kingma & Welling, 2013) amortize inference with $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ as variational posterior (encoder) and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ as likelihood (decoder). The encoder and decoder are parameterized by neural networks with parameters ϕ, θ. Nets are optimized simultaneously via maximisation of the evidence lower bound (ELBO) objective:

$$\mathcal{L}(\theta, \phi, \lambda) \triangleq \mathbb{E}_{\substack{p_e(\boldsymbol{x}) \\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \left(\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})\right) \leq \mathbb{E}_{p_e(\boldsymbol{x})}\left[\log \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}\right]. \tag{1}$$

Typically, instead of the density $p_e(\boldsymbol{x})$ we are given a dataset $\{\boldsymbol{x}_n\}_{n=1}^N$ and consider an empirical distribution $\frac{1}{N}\sum_{n=1}^N \delta_{\boldsymbol{x}_n}(\boldsymbol{x})$.

Optimal Prior and Aggregated Posterior Hoffman & Johnson (2016); Goyal et al. (2017) discuss the choice of the prior distribution over latent space $\pi(\boldsymbol{z})$ and observe that default choice of the Gaussian prior significantly restricts the expressiveness of the model. Tomczak & Welling (2018) proposes to use a prior that optimizes the ELBO(1). The solution of this problem is the aggregated variational posterior:

$$\hat{q}(\boldsymbol{z}) = \mathbb{E}_{p_e(\boldsymbol{x})} q_\phi(\boldsymbol{z}|\boldsymbol{x}). \tag{2}$$

For an empirical $p_e(\boldsymbol{x})$, $\hat{q}(\boldsymbol{z}) = \frac{1}{N}\sum_{n=1}^N q_\phi(\boldsymbol{z}|\boldsymbol{x}_n)$ and Tomczak & Welling (2018) proposes a parametric approximation to avoid over-fitting and reduce computational costs: $\pi(\boldsymbol{z}) = \frac{1}{K}\sum_{k=1}^K q_\phi(\boldsymbol{z}|\boldsymbol{u}_k)$. Parameter $\{\boldsymbol{u}_k\}_{k=1}^K \in \mathbb{R}^{K \times D}$ is optimized simultaneously with (θ, ϕ) by the ELBO(1) maximization, and K is a hyper-parameter of the algorithm. The additive structure of the optimal prior $\pi^*(\boldsymbol{z})$ over the dataset points motivates us to consider such a prior distribution for continual learning.

Continual learning framework In the continual learning framework, we do not have access to the whole dataset. We define the sequence of tasks to be solved $t=1,\dots,T$. Subsets of the data for each task $\mathcal{D}^1,\dots,\mathcal{D}^T$ arrive sequentially and we have access only to the data of the current task. For each task t, we consider the empirical distribution $p_e^t(\boldsymbol{x})=\frac{1}{|\mathcal{D}^t|}\sum\limits_{\boldsymbol{x}_n\in\mathcal{D}^t}\delta_{\boldsymbol{x}_n}(\boldsymbol{x})$.

3 BooVAE Algorithm (Proposed Method)

We start from defining an optimal prior for the VAE model in the continual learning framework. Next, we find the optimal additive expansion of the current prior to match the innovation coming from the new task. We use obtained result and provide a general algorithm for training the VAE model in the continual learning framework. It works as an iterative Minorization-Maximization algorithm. In the minorization step, we expand the current approximation of the prior and learn pseudo-inputs. At the Maximization step, we update parameters of the encoder and the decoder with the prior being fixed.

3.1 Optimal prior in continual learning

In this section, we derive an optimal prior for the VAE model in the continual framework and provide the algorithm to it approximation. We provide skipped technical details in Sup.(A.1). We start from the following decomposition of the ELBO(1):

$$\mathcal{L}(\theta,\phi,\pi)=\mathbb{E}_{p_e(\boldsymbol{x})}[\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\log p_\theta(\boldsymbol{x}|\boldsymbol{z})-(\mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})|\hat{q}(\boldsymbol{z})]+\mathrm{KL}[\hat{q}(\boldsymbol{z})|\pi(\boldsymbol{z})])], \tag{3}$$

where $\hat{q}(\boldsymbol{z})=\mathbb{E}_{p_e(\boldsymbol{x})}q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is the aggregated variational posterior. As KL-divergence is non-negative, the global maximum of Eq.(3) over π is reached when:

$$\pi^*(\boldsymbol{z})=\hat{q}(\boldsymbol{z})=\mathbb{E}_{p_e(\boldsymbol{x})}q_\phi(\boldsymbol{z}|\boldsymbol{x}). \tag{4}$$

We assume that for the sequence of the two tasks, we can write the data distribution as the discrete mixture of two distributions: $p_e(\boldsymbol{x})=\alpha\, p_e^1(\boldsymbol{x})+(1-\alpha)\, p_e^2(\boldsymbol{x}), \alpha\in(0;1)$. This assumption holds for empirical data distribution, which is of the most interest for us: $p_e(\boldsymbol{x})=\frac{1}{N}\sum_{n=1}^N\delta_{\boldsymbol{x}_n}(\boldsymbol{x})=\frac{|\mathcal{D}^1|}{|\mathcal{D}^1|+|\mathcal{D}^2|}p_e^1(\boldsymbol{x})+\frac{|\mathcal{D}^2|}{|\mathcal{D}^1|+|\mathcal{D}^2|}p_e^2(\boldsymbol{x})$. Then we can decompose the ELBO(1) as following:

$$\mathcal{L}(\theta,\phi,\pi)=\mathbb{E}_{\substack{p_e(\boldsymbol{x})\\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}}\log p_\theta(\boldsymbol{x}|\boldsymbol{z})-\sum_{i=1,2}\alpha_i(\mathbb{E}_{p_e^i(\boldsymbol{x})}\mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})|\hat{q}_i(\boldsymbol{z})]+\mathrm{KL}[\hat{q}_i(\boldsymbol{z})|\pi(\boldsymbol{z})]), \tag{5}$$

where $\alpha_1=\alpha, \alpha_2=1-\alpha$ and $\hat{q}_i(\boldsymbol{z})=\mathbb{E}_{p_e^i(\boldsymbol{x})}q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Keeping only terms with dependence over the prior distribution, we conclude to the optimization problem over the probability density space $\mathcal{P}$:

$$\min_{\pi(\boldsymbol{z})\in\mathcal{P}}\alpha\,\mathrm{KL}[\hat{q}_1(\boldsymbol{z})|\pi(\boldsymbol{z})]+(1-\alpha)\,\mathrm{KL}[\hat{q}_2(\boldsymbol{z})|\pi(\boldsymbol{z})]. \tag{6}$$

Hence, if we can store the data for both tasks we can use the same optimal prior $\mathbb{E}_{\boldsymbol{x}}\, q_\phi(\boldsymbol{z}|\boldsymbol{x})$, where expectation is taken with respect to $\alpha\, p_e^1(\boldsymbol{x})+(1-\alpha)\, p_e^2(\boldsymbol{x})$. However, in continual setting we don't have access to the $p_e^1(\boldsymbol{x})$, neither it is reasonable to store $\hat{q}_1(\boldsymbol{z})$. Thus after Task 1, we consider using the approximation of the aggregated variational posterior of the first task $\hat{q}_1^a(\boldsymbol{z})\approx\hat{q}_1(\boldsymbol{z})$. This modification of the problem (6) leads to the following prior:

$$\pi^{1,2}(\boldsymbol{z})=\alpha\hat{q}_1^a(\boldsymbol{z})+(1-\alpha)\hat{q}_2(\boldsymbol{z})=\arg\min_{\pi(\boldsymbol{z})\in\mathcal{P}}\alpha\,\mathrm{KL}[\hat{q}_a(\boldsymbol{z})|\pi(\boldsymbol{z})]+(1-\alpha)\,\mathrm{KL}[\hat{q}_2(\boldsymbol{z})|\pi(\boldsymbol{z})]. \tag{7}$$

Next, we consider that the optimal prior for the first task $\pi^1(\boldsymbol{z})$ is a good start for a new prior after observing the new task. Hence, we propose using an additive expansions of the current prior to match innovation induced by the new task. Then the objective of the problem (7) transforms to:

$$\min_{h\in\mathcal{P}}\alpha\,\mathrm{KL}[\hat{q}_1^a(\boldsymbol{z})|(1-\beta)\pi^1(\boldsymbol{z})+\beta h(\boldsymbol{z})]+(1-\alpha)\,\mathrm{KL}[\hat{q}_2(\boldsymbol{z})|(1-\beta)\pi^1(\boldsymbol{z})+\beta h(\boldsymbol{z})]). \tag{8}$$

The optimization problem (8) is bi-convex over h and β. We consider the Functional Frank-Wolfe algorithm (informally, "boosting") (Wang et al., 2015) for the corresponded objective:

$$\begin{aligned}&\mathcal{F}_{\alpha,\beta}[\hat{q}_1^a(\boldsymbol{z}),\hat{q}_2(\boldsymbol{z});\pi^1(\boldsymbol{z}),h(\boldsymbol{z})]=\\&=\alpha\,\mathrm{KL}[\hat{q}_1^a(\boldsymbol{z})|(1-\beta)\pi^1(\boldsymbol{z})+\beta h(\boldsymbol{z})]+(1-\alpha)\,\mathrm{KL}[\hat{q}_2(\boldsymbol{z})|(1-\beta)\pi^1(\boldsymbol{z})+\beta h(\boldsymbol{z})]).\end{aligned} \tag{9}$$

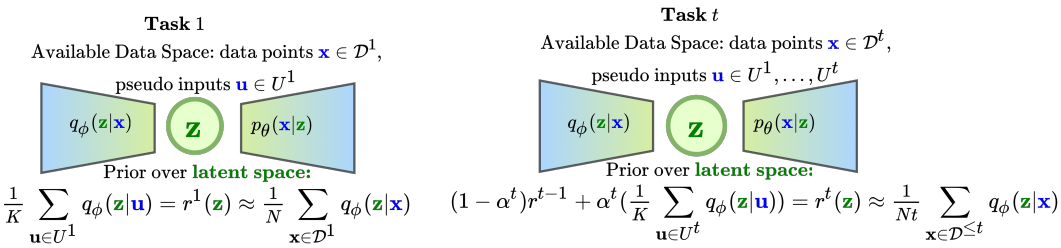


Figure 1: We propose to expand prior distribution in order to match new information from the coming task. We parametrize each component in the prior distribution with encoder, evaluated on the trainable pseudo-inputs. These pseudo inputs store information about the data at the correspondent task.

We aim to find h as the projection of the functional gradient to the probability density space $\mathcal{P}$ and then optimize over β with the stochastic gradient method, while h fixed. The functional gradient of the objective (8) with respect to perturbation h is $\alpha\frac{\hat{q}_1^a(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}+(1-\alpha)\frac{\hat{q}_2(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}$ in the following sense:

$$\begin{aligned}&\mathcal{F}_{\alpha,\beta}[\hat{q}_1^a(\boldsymbol{z}),\hat{q}_2(\boldsymbol{z});\pi^1(\boldsymbol{z}),h(\boldsymbol{z})]=\\&=\mathcal{F}_\alpha[\hat{q}_1^a(\boldsymbol{z}),\hat{q}_2(\boldsymbol{z});\pi^1(\boldsymbol{z})]-\beta\left(\int d\boldsymbol{z}\; h(\boldsymbol{z})\left[\alpha\frac{\hat{q}_1^a(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}+(1-\alpha)\frac{\hat{q}_2(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}\right]-1\right)+o(\beta).\end{aligned} \tag{10}$$

We consider projection to the probability space as the solution of the optimization problem:

$$\max_{h\in\mathcal{P}}\log\int h(\boldsymbol{z})\left[\alpha\frac{\hat{q}_1^a(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}+(1-\alpha)\frac{\hat{q}_2(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}\right]d\boldsymbol{z}\geq\max_{h\in\mathcal{P}}\int h(\boldsymbol{z})\log\left[\alpha\frac{\hat{q}_1^a(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}+(1-\alpha)\frac{\hat{q}_2(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}\right]d\boldsymbol{z}. \tag{11}$$

We use the lower bound in the (11) in order to use Monte-Carlo estimates of the gradients. The problem is linear over $h(\boldsymbol{z})$ and the solution is degenerate distribution. To this end, we add the entropy regularization $H[h]=-\int h(\boldsymbol{z})\log h(\boldsymbol{z})\,d\boldsymbol{z}$ and obtain the final objective:

$$\min_{h\in\mathcal{P}}\mathrm{KL}\left[h\middle|\frac{\alpha\hat{q}_1^a(\boldsymbol{z})+(1-\alpha)\hat{q}_2(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}\right]. \tag{12}$$

As far as we update the initial prior $r^0(\boldsymbol{z})=\pi^1(\boldsymbol{z})$ with the mixture component h: $r^1(\boldsymbol{z})=(1-\beta)\pi^1(\boldsymbol{z})+\beta h(\boldsymbol{z})$, we can improve current approximation by finding the new component, solving the same optimization problem, but using r^1 instead of π^1. The objective (12) encourages a new component to be different from the already constructed approximation:

$$\mathrm{KL}\left[h\middle|\frac{\alpha\hat{q}_1^a(\boldsymbol{z})+(1-\alpha)\hat{q}_2(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}\right]=\underbrace{\mathrm{KL}\left[h|\alpha\hat{q}_1^a(\boldsymbol{z})+(1-\alpha)\hat{q}_2(\boldsymbol{z})\right]}_{\text{match the target prior}}-\underbrace{\int h(\boldsymbol{z})\log\frac{1}{\pi^1(\boldsymbol{z})}d\boldsymbol{z}}_{\substack{\text{be different}\\ \text{from the current approximation}}}. \tag{13}$$

To this end, we have defined the algorithm of learning optimal prior for VAE in continual setting. Now we will use it to formulate the algorithm of VAE training in continual learning framework.

3.2 BooVAE Algorithm

In this section, we formulate the algorithm for continual learning – BooVAE (short for Boosting VAE). It works as the iterative Minorization-Maximization algorithm. In the *minorization step*, we use the obtained optimization problem (12) to learn a new component of the prior. These steps are alternated with *ELBO maximization steps* until the desired number of components in prior is reached. From that point further on, only model parameters are updated until convergence. As at maximization step follows the usual routine used in training VAEs, our approach can be easily used on top of any VAE model. The derivations of applications of BooVAE to the VAE with flow-based prior presented at Sup.(A.2). These steps are summarized in Alg.(1) and idea of prior learning for continual setting is illustrated in Fig.(1). Now, we are going to consider them in more details.

Prior Update Step At this step we expand the current approximation of the optimal prior distribution. Subsets $\mathcal{D}^1, \dots, \mathcal{D}^T$ arrive sequentially and may come from different domains. We have access only to one subset of data and current prior at a time but aim at learning the prior distribution $\frac{1}{|\mathcal{D}^{\leq T}|}\sum_{\boldsymbol{x}\in\mathcal{D}^{\leq T}} q_\phi(\boldsymbol{z}|\boldsymbol{x})$. At the task t we start from the current prior $r^t(\boldsymbol{z}) := r^{t-1}(\boldsymbol{z})$. We expand current approximation with at most K components. At each step k we add a new component h to the current approximation r^t to move it towards an optimal prior $\pi^{\leq t}$. Following (Tomczak & Welling, 2018), we select the parametric family $\mathcal{R}$ of the components h as learnable pseudo-inputs $\boldsymbol{u}$ to the encoder: $h(\boldsymbol{z}) = q_\phi(\boldsymbol{z}|\boldsymbol{u})$. This choice connects parameters of the prior with the data-space. We conclude with a two step procedure for adding a new component:

1. Train new component:
   $h^* = \arg\min_{h\in\mathcal{R}} \mathrm{KL}\left[h \middle| \frac{\pi^{\leq t}}{r^t}\right]$.
2. Train component weight:
   $\beta^* = \arg\min_{\beta\in(0;1)} \mathrm{KL}\left[\beta h^* + (1-\beta) r^t \middle| \pi^{\leq t}\right]$.

Algorithm 1 BooVAE algorithm

input Current task t dataset $\mathcal{D}$: $\{x_n\}_{n=1}^N$
input Maximal number of components K
  Prior to approximate $\pi^{\leq t} = \alpha r^{t-1} + (1-\alpha)\hat{q}_t$.
  Initialize prior $r^t = r^{t-1}$
  $\theta^*, \phi^* = \arg\max_{\theta,\phi} \mathcal{L}(r^t, \theta, \phi)$
  $k = 1$
  while not converged and $k < K$ **do**
    $h^* = \arg\min_{h\in\mathcal{R}} \mathrm{KL}\left[h \middle| \frac{\pi_{\lambda^*}}{r^t}\right]$
    $\beta^* = \arg\min_{\beta\in(0;1)} \mathrm{KL}[\beta h + (1-\beta) r^t | \pi^{\leq t}]$
    $r^t = \beta^* h^* + (1-\beta^*) r^t$
    $k = k+1$
    $\theta^*, \phi^* = \arg\max_{\theta,\phi} \mathcal{L}(r^t, \theta, \phi)$
  end while
output r^t, θ^*, ϕ^*

As it was already mentioned, we define optimal prior to be equal to aggregate posterior. Therefore, it stores all the information about training dataset. The optimal prior for tasks $1:t$ can be expressed with prior from the previous step (e.g. aggregate posterior over tasks $1:t-1$) and training dataset from the current task only:

$$\pi^{\leq t}(\boldsymbol{z}) = \frac{|\mathcal{D}^{\leq t-1}|}{|\mathcal{D}^{\leq t}|}\pi^{\leq t-1}(\boldsymbol{z}) + \frac{|\mathcal{D}^t|}{|\mathcal{D}^{\leq t}|}\frac{1}{|\mathcal{D}^t|}\sum_{\boldsymbol{x_n}\in\mathcal{D}^t} q_\phi(\boldsymbol{z}|\boldsymbol{x_n}). \tag{14}$$

Since we don't have access to the data from the previous tasks, we suggest using trained prior r^{t-1} as a proxy for the corresponding part of the mixture. We use random subset from the current task $\mathcal{M}^t \subset \mathcal{D}^t$ containing N_t observations to estimate aggregate posterior of the current task.

$$\pi^{\leq t} \approx \frac{|\mathcal{D}^{\leq t-1}|}{|\mathcal{D}^{\leq t}|} r^{t-1}(\boldsymbol{z}|\{\boldsymbol{u}\}^{\leq t-1}) + \frac{|\mathcal{D}^t|}{|\mathcal{D}^{\leq t}|}\frac{1}{N_t}\sum_{\boldsymbol{x}\in\mathcal{M}^t} q_\phi(\boldsymbol{z}|\boldsymbol{x}). \tag{15}$$

Such formulation allows an algorithm not to forget information from the previous task, which is stored in the prior distribution and pseudo-inputs $\{\boldsymbol{u}\}^{\leq t-1}$. The prior distribution performs regularization for the VAE model. As the budget of components learning per-task is reached, we perform component pruning by weights optimization, see Supp.(A.3).

ELBO Maximization Step. At this step parameters of the encoder and decoder θ, ϕ are updated. We add regularization term, to ensure that the model remembers the information, which is stored in the pseudo-inputs. Given the fixed mean-parameters of prior distribution from the previous task r^{t-1}, we keep its components during training the task t as well as distribution of their fixed decoded mean-parameters $p^{t-1}(\cdot)$:

$$R_{enc}(\phi) = \sum_{\boldsymbol{u}\in r^{t-1}} \mathrm{KL}_{\text{sym}}\left[q_\phi(\boldsymbol{z}|\boldsymbol{u}) | r^{t-1}(\boldsymbol{z}|\boldsymbol{u})\right], R_{dec}(\theta) = \sum_{\boldsymbol{z}\sim r^{t-1}} \mathrm{KL}_{\text{sym}}\left[p_\theta(\boldsymbol{x}|\boldsymbol{z}) | p^{t-1}(\boldsymbol{x}|\boldsymbol{z})\right]. \tag{16}$$

where $\mathrm{KL}_{\text{sym}}(p|q) = \frac{1}{2}\left(\mathrm{KL}(p|q) + \mathrm{KL}(q|p)\right)$. We use symmetric KL-divergence, since we have observed that it help to avoid encoder and decoder to memorizing delta-function centered in the pseudo-input and results in more diverse samples from the previous tasks. Final objective for the maximization step over ϕ, θ for the task t is the following:

$$\mathbb{E}_{\substack{\boldsymbol{x}\sim\mathcal{D}_t,\\ \boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})}} \left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) - \mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x}) | r^t(\boldsymbol{z})]\right] - \lambda(R_{enc}(\phi) + R_{dec}(\theta)). \tag{17}$$

4 Related Work

Boosting density approximation. The approximation of the unnormalized distribution with the sequential mixture models has been considered previously in several studies. Several works (Miller et al., 2017; Gershman et al., 2012) perform direct optimization of ELBO with respect to the new component's parameters. Unfortunately, it leads to unstable optimization problem. Therefore, other works consider functional Frank-Wolfe framework, where subproblems are linearized (Wang et al., 2015). At each step, KL-divergence functional is linearized at the current approximation point by its convex perturbation, obtaining tractable optimization subproblems over distribution space for each component. Guo et al. (2016) suggested using concave log-det regularization for gaussian base learners. Egorov et al. (2019), Locatello et al. (2018) used entropy-based regularization. In our work, we come up with a similar optimization algorithm. We optimize *different objective for different propose*: the weighted sum of exclusive KL to approximate optimal prior for VAE, while mentioned works approximate inclusive KL, in order to approximate posterior distriubtion.

Continual learning for VAE. (Lesort et al., 2019) compare EwC, generative replay and random coresets. We use a single pair of encoder and decoder for all the tasks unless otherwise stated. Nguyen et al. (2018) propose to use multi-head VAE architecture, where a separate encoder is trained for each task, and an extra head is added to the decoder. We believe that even though this model was shown to produce good results, it has several crucial limitations. Firstly, it makes scalability of the method very poor, since the number of parameters in the models increases dramatically with the number of tasks. Secondly, it requires task labels to be known both during training and validation to use the suitable "head" for each data point. Finally, it limits the amount of information we can share between tasks and thus may lead to worse performance for the tasks, which has fewer data available and have a lot of similarities with other tasks. We have observed the latter phenomenon on the CelebA dataset, where all the tasks are supposed to generate images with faces but with different hair color. Recent work (Rao et al., 2019) use learnable mixture as prior distribution. However, this reached by also expanding the architecture of encoder with multi-heads and using self-replay to overcome forgetting. Achille et al. propose similar ideas of encoder expansion and use also use self-replay. However, our goal is to provide orthogonal approach, which avoids self-replay and multi-heads, but can be combined with them.

5 Experiments

We empirically evaluation our algorithm on both disjoint image generation and classification tasks. In the latter case we suggest using VAE learned in the continual setting for generative replay (Shin et al., 2017).

5.1 Disjoint image generation task

Setup. We consider an experimental setup, where each task contains objects of 1 class (e.g. one digit for MNIST dataset). The resulting generative model, trained on all the classes one-by-one is supposed to generate images from all the classes. We perform experiments on MNIST, notMNIST, fashion MNIST and CelebA datasets. Each of MNIST datasets has 10 classes with digits, characters and pieces of clothing correspondingly. For the CelebA dataset we consider 4 tasks based on the attributes: black, blond, brown and gray hair. For all the datasets we use VAE (Kingma & Welling, 2013) model with gaussian posterior. A complete description of the experimental setup, architectures and hyperparameters can be found in Supp. (B.5).

Compared methods. As a baseline we consider standard VAE model. We expect it to suffer from catastrophic forgetting and use it as a lower bound on the model performance. We refer to this model as **Standard**. Our method has some similarities with the regularization-based continual learning algorithms, which add quadratic penalty on the weights. The difference is that we use prior, defined as a mixture of the encoded pseudo-inputs, to avoid catastrophic forgetting both for the encoder and the decoder, Eq. (16). In the experiments we compare with such regularization-based approaches as Elastic Weight Consolidation (**EWC**) (Kirkpatrick et al., 2017) and Variational Continual Learning (**VCL**) (Nguyen et al., 2018). Our approach uses pseudo-inputs to approximate an optimal prior distribution for each task. This procedure can also be seen as coreset selection (Huggins et al., 2016;

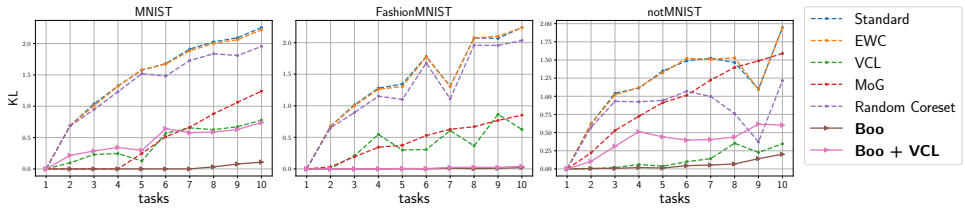


Figure 2: **Diversity** of generated samples, estimated as a KL between discrete distribution with the equal probability for each class and empirical distribution of samples from VAEs. The lower is the better. We conclude that BooVAE outperforms other approaches.

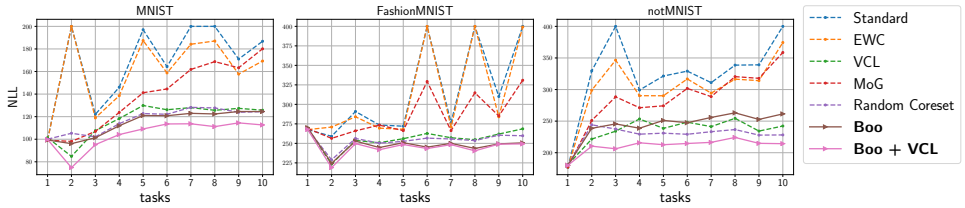


Figure 3: **NLL** on the full test dataset averaged over 5 runs after continually training on 10 tasks. Lower is better. We observe that BooVAE performance is comparable or better than of other methods.

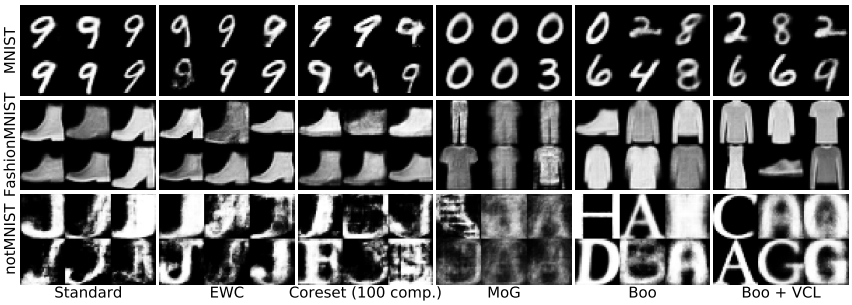


Figure 4: Samples from the VAEs after the last task is learned. BooVAE keep sampling different tasks, while other approaches suffer from catastrophic forgetting.

Bachem et al., 2015). To validate the quality of the learned coresets, we perform the comparison with **random coresets**, which was also used in (Nguyen et al., 2018). We store a random subsample of the training dataset from previous tasks and add them to each batch during training to avoid catastrophic forgetting. Moreover, we can learn the prior distribution components in the latent space instead of the data space as we do with the pseudo-inputs. We have conducted experiments, where the prior is trained as a mixture of Gaussian's (with diagonal covariance) in the latent space. We refer to this approach as **MoG**. Just as we do in BooVAE, in MoG we learn a new mixture of components for each task and use the regularization from Eq.(16) to avoid catastrophic forgetting.

Metrics. To evaluate the performance of the VAE approach on grey scale images, we a standard metric – negative log-likelihood (**NLL**) on the test set. NLL is calculated by importance sampling

Table 1: FID values for CelebA dataset averaged over 5 runs, the lower is the better. Each row corresponds to the FID for cumulative learned tasks of different hair types. BooVAE outperform other approaches, including multihead.

#T	EWC	Multihead + EWC	VCL	Multihead + VCL	Random Coreset (40)	Random Coreset (80)	Boo (40 comp.)
1	35.8 (0.8)	37.7 (1.1)	35.8 (0.6)	37.93 (1.0)	36.4 (1.3)	38.1 (1.9)	**27.5 (0.6)**
2	61.4 (1.7)	69.2 (1.3)	58.7 (0.6)	65.54 (0.2)	59.6 (0.4)	59.8 (0.8)	**45.7 (2.4)**
3	40.7 (0.6)	45.2 (0.8)	39.3 (0.6)	42.18 (0.4)	39.7 (0.8)	41.9 (0.1)	**33.4 (1.4)**
4	50.7 (0.6)	75.7 (0.8)	48.7 (0.9)	75.16 (2.0)	48.1 (1.1)	47.4 (1.9)	**43.1 (0.2)**

Figure 5: Samples from VAE trained on CelebA. For each model rows correspond to cumulatively learned tasks of different hair types.

method, using 5000 samples for each test observation:

$$-\log p(x) \approx -\log \frac{1}{K} \sum_{i=1}^{K} \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z}_i)p(\boldsymbol{z}_i)}{q_\phi(\boldsymbol{z}_i|\boldsymbol{x})},\ \boldsymbol{z}_i \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}).$$

NLL measures the quality of the reconstructions that VAE produces. It is also essential to evaluate the diversity of generated images with respect to all the tasks in the continual setting. In our experiments, each task contains a new class. Thus, we expect our model to generate images from all the classes $t{=}1, \dots, T$ in the same proportion as they appear in the training dataset. For the dataset with balanced classes, this proportion is equal to $\frac{1}{T}$. We assess the diversity using the sum of **KL-divergences** between T pairs of Bernoulli distributions:

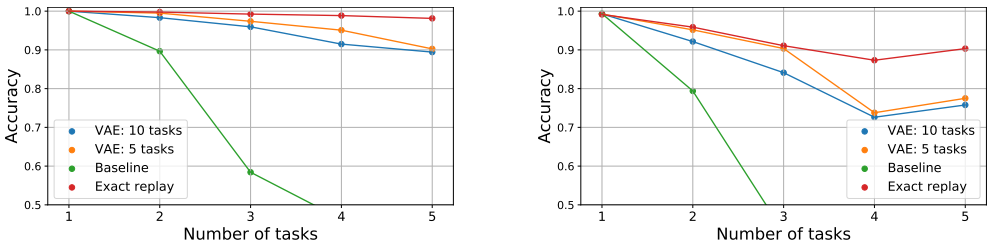
$$\sum_{t=1}^{T} \mathrm{KL}\left[p_t || \widehat{p}_t\right],\ p_t \sim \mathrm{Be}\left(\tfrac{1}{T}\right),\ \widehat{p}_t \sim \mathrm{Be}\left(\tfrac{N_t}{\sum_{t=1}^{T} N_t}\right),$$

where $\widehat{p}_t$ is an empirical distribution of the generated classes, N_t — number of generated images from the class t. To estimate N_t, we train the classification neural network to achieve high accuracy and use it to classify images generated by the model. We use 10^4 samples in total to calculate the metric. For the CelebA dataset, we used Frechet Inception Distance (**FID**) (Heusel et al., 2017) over 10^4 samples, which is supposed to measure both quality and diversity of generated samples. FID rely heavily on the implementation of Inception network (Barratt & Sharma, 2018); we use PyTorch version of the Inception V3 network (Paszke et al., 2017).

Results on MNIST(s). In Fig.(2),(3) we provide results for three MNIST datasets. Both figures depict values averaged over five runs. We report mean values, standard deviations and comparison with the multihead architectures in Supp.(B.1). We evaluate the performance of VAE on the test dataset after each new task is added. The x-axis denotes a total number of tasks seen by the models (and thus, the total number of tasks in the test dataset). The flatter the line is, the less forgetting we observe as a number of tasks increases. We provide numbers for each task separately in Supp.(B.2). Notice that BooVAE can be combined with any other weight-regularization method. We have observed that EWC does not improve performance a lot (see Supp.(B.1)), therefore we report only results for the combination of BooVAE with VCL, which gives the best performance in terms of NLL.

We observe that, based on KL metric, pure BooVAE produces the most diverse samples. We plot several samples from the model after training it on ten tasks in Fig.(4). For MoG, Random Corset and BooVAE, we use the same amount of components equal to 15 for each task. It is worth mentioning that the closer the random corset size is to the size of the training data, the better the performance should be. In the limit case, we can store all the data from the previous tasks, guaranteeing the absence of catastrophic forgetting. Our goal is to reduce the amount of stored information, thus we used a pretty small number of components. In Supp.(B.4) we explore, how large the random corset should be to get the performance comparable to BooVAE.

Results on CelebA We conduct experiments on CelebA dataset for several reasons. Firstly, we want to show that our method works not only on small-scale images, such as MNIST but also on higher-dimensional data. Secondly, since the classes differ only by the hair color in this setting, we may see the advantages of the shared architecture more clearly. In the CelebA dataset there are much fewer faces with grey hair, compared to other colors. Therefore, information from other



(a) MNIST (b) Fashion MNIST

Figure 6: We consider continual learning experiment for MNIST and Fashion MNIST datasets, where we split the dataset in 5 task, containing pairs of disjoint classes, i.e. '0/1', '2/3', etc. We train both BooVAE and classification DNN in continual setting, using VAE for generative replay to avoid catastrophic forgetting in classification.

classes is essential to obtain good results on these images. We compare the FID values as new tasks are added to the models in Table(1). The BooVAE outperforms other approaches, including model with multihead architecture. Notice that multihead fails on the last task, which has much fewer observations, compared to others. This highlights the benefit of the shared architecture, as the Multi-heads approach limits the amount of shared information between tasks. Samples from the different VAEs can be found on Fig.(5). We provide more samples in Supp.(B.3).

5.2 Generative Replay for Discriminative Model with continual VAE

Motivation Common approach to mitigate catastrophic forgetting in discriminative models is deep generative replay (Shin et al., 2017). The method is based on the recollection of the past knowledge, such as the data of past classes, by generating it from the trained generative model. However, since continual learning for generative models was limited, it was proposed to re-train the generative model from scratch when a new task arrives. Since we propose the method for the continual learning of the generative model, we can avoid this. We can continually train VAE along with the discriminative model.

Setting We perform the continual learning experiment using the MNIST and Fashion MNIST datasets. We apply splitMNIST setting when the dataset is split into five binary classification tasks. E.g. the first task containing digits '0' and '1', the second task — digits '2' and '3', and so on. We perform similar splitting into the five tasks for the Fashion MNIST. We train MLP with two hidden layers, LeakyReLU activations and Dropout layers. For each task, we add a new classification head (one fully connected layer) and train for 200 epochs with batch size 500.

Results The mean accuracy across all tasks is reported in Fig.(6). We provide two models for comparison. Exact replay setting reuses all the training data from the previous task, thus, giving an upper bound on the generative replay's performance. In the baseline method, we do not use any information about the previous task. This gives us a lower bound on the performance, i.e. if replay buffer is not used. We report two versions of the generative replay with VAE. In the first one, we consider each class as a separate task and train VAE in the continual setting (VAE: 10 tasks) and sample images from the last available model. We use prior components to label the image classes. For example, along with the first classification task on MNIST (with digits '0' and '1') we train VAE firstly on '0's and then on '1's. As a result, we have separate components in the prior distribution for both tasks. Sampling latent vector from these components decoding them gives us the replay buffer for the next classification task. In the second setting, we combine classes like in splitMNIST setting, i.e. each task contains two classes (VAE: 5 tasks). In this setting, samples from the prior have to be classified in order to be used in classification model. We follow the approach from Shin et al. (2017), using classier from the previous step for this purpose. Results that we observe are comparable with the MNIST results in Shin et al. (2017)[3], while our approach avoids full re-training of the generative model.

[3]Authors provide only plot (Fig. 2 in their paper), therefore we are not able to report exact numbers for comparison

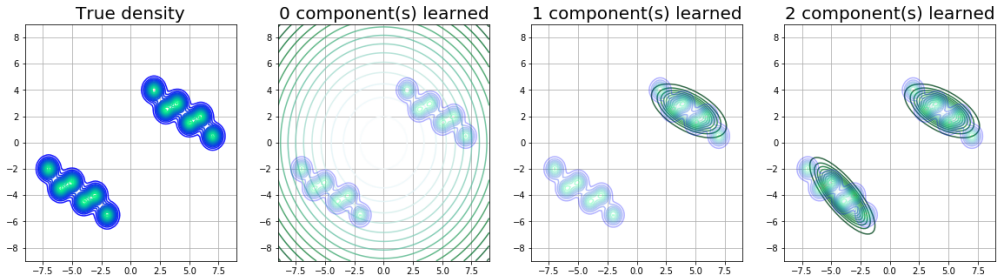


Figure 7: The mixture of 12 Gaussian is approximated by 2-component Gaussian mixture with Eq.(11). This toy experiment illustrates the intuition of our approach, where we hope to efficiently approximate prior $\pi^*(\boldsymbol{z}) = \mathbb{E}_{p_e(\boldsymbol{x})} q_\phi(\boldsymbol{z}|\boldsymbol{x})$ given only empirical version of the target density $\frac{1}{N}\sum_{n=1}^{N} q_\phi(\boldsymbol{z}|\boldsymbol{x}_n)$.

6 Conclusion

We propose a novel algorithm for continual learning of VAEs with the static architecture by incorporating the data-driven information about new task into the prior over the latent space. We leverage the specific structure of the VAE model and match new data innovation with the additive aggregated posterior expansion. The boosting-like approach allows us to reduce the number of components in the approximation of the optimal prior distribution without the loss of performance. We empirically validate performance of our algorithm and compare with other approaches. That being said, the proposed algorithm is orthogonal to them and could be easily combined. We would like to finalize with additional comments on performance evaluation and approach intuition.

Model perfomance In the continual learning setting it is important to evaluate the evolution of the metrics for each task while new task arrives. Hence in Sections (B.1) - (B.3) we provide more detailed overview of models performance and report NLL and diversity metric after each additional task is trained. We discuss how the performance changes on the whole test dataset as we keep training in continual learning setting. Moreover, we report and discuss these metrics for each task separately in Section (B.2). Finally, we visually study the samples from the model on each step in Section (B.3).

Approach intuition We provide toy-example visualization in Fig.(8,7) to illustrate equations at Sec.(3).

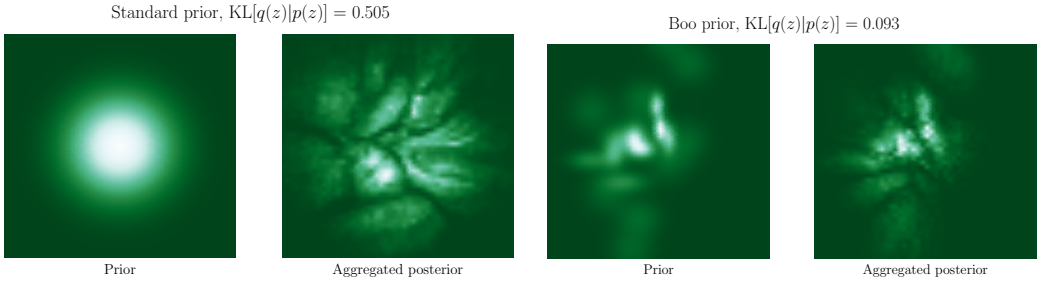


Figure 8: Visualization of the prior density and aggregate posterior for the VAE with 2-d latent space. In left we present results for the VAE with $\mathcal{N}(\boldsymbol{z}|\mathbf{0}, I)$ prior and in the right with the proposed Boo prior. By visual inspection and KL-divergence comparison we conclude that the proposed prior matches the aggregated posterior better that a standard normal prior.

Acknowledgments

Authors are thankful to Jakub Tomczak for feedback and inspiring words, to Kirill Neklyudov and Dmitry Molchanov for important discussions during 2019 and to the reviewers who provided detailed feedback. The work of Evgeny Burnaev was supported by Ministry of Science and Higher Education grant No. 075-10-2021-068.

References

Achille, A., Eccles, T., Matthey, L., Burgess, C. P., Watters, N., Lerchner, A., Higgins, I., and UCLA, D. Life-long disentangled representation learning with cross-domain latent homologies.

Bachem, O., Lucic, M., and Krause, A. Coresets for nonparametric estimation-the case of dp-means. In *International Conference on Machine Learning*, pp. 209–217. PMLR, 2015.

Barratt, S. and Sharma, R. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

Benjamin, A., Rolnick, D., and Kording, K. Measuring and regularizing networks in function space. In *International Conference on Learning Representations*, 2018.

Egorov, E., Neklydov, K., Kostoev, R., and Burnaev, E. Maxentropy pursuit variational inference. In *International Symposium on Neural Networks*, pp. 409–417. Springer, 2019.

Gershman, S. J., Hoffman, M. D., and Blei, D. M. Nonparametric variational inference. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pp. 235–242. Omnipress, 2012.

Goyal, P., Hu, Z., Liang, X., Wang, C., and Xing, E. P. Nonparametric variational auto-encoders for hierarchical representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5094–5102, 2017.

Guo, F., Wang, X., Fan, K., Broderick, T., and Dunson, D. B. Boosting variational inference. *arXiv preprint arXiv:1611.05559*, 2016.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pp. 6626–6637, 2017.

Hoffman, M. D. and Johnson, M. J. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, volume 1, pp. 2, 2016.

Huggins, J. H., Campbell, T., and Broderick, T. Coresets for scalable bayesian logistic regression. *arXiv preprint arXiv:1605.06423*, 2016.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

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

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

Lee, W., Song, K., and Moon, I.-C. Augmented variational autoencoders for collaborative filtering with auxiliary information. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1139–1148, 2017.

Lesort, T., Caselles-Dupré, H., Garcia-Ortiz, M., Stoian, A., and Filliat, D. Generative models from the perspective of continual learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2019.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2018.

Liu, X., Masana, M., Herranz, L., Van de Weijer, J., Lopez, A. M., and Bagdanov, A. D. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 2262–2268. IEEE, 2018.

Locatello, F., Khanna, R., Ghosh, J., and Ratsch, G. Boosting variational inference: an optimization perspective. In *International Conference on Artificial Intelligence and Statistics*, pp. 464–472. PMLR, 2018.

Miller, A. C., Foti, N. J., and Adams, R. P. Variational boosting: Iteratively refining posterior approximations. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2420–2429. JMLR. org, 2017.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. In *International Conference on Learning Representations*, 2018.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Rao, D., Visin, F., Rusu, A., Pascanu, R., Teh, Y. W., and Hadsell, R. Continual unsupervised representation learning. *Advances in Neural Information Processing Systems*, 32:7647–7657, 2019.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

Tomczak, J. and Welling, M. Vae with a vampprior. In *International Conference on Artificial Intelligence and Statistics*, pp. 1214–1223. PMLR, 2018.

Wang, C., Wang, Y., Schapire, R., et al. Functional frank-wolfe boosting for general loss functions. *arXiv preprint arXiv:1510.02558*, 2015.

Zhou, L., Deng, W., and Wu, X. Unsupervised anomaly localization using vae and beta-vae. *arXiv preprint arXiv:2005.10686*, 2020.

[Yes] , [No] , or [N/A]

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Limitations are illustrated in Sec.5 and Sup.B.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Our method is general algorithm for VAE continual learning
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Please see Sup.A
   (b) Did you include complete proofs of all theoretical results? [Yes] Please see Sup.A
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide code and readme in attached .zip
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide all hyperparameters and details of the experiments in Sup.B,B.5
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We provide error bars and number of runs in the main text and in Sup.B.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please see Sup.B.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] We provide only code written by ourselves.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Supplementary Material
BooVAE: Boosting Approach for Continual Learning of VAE

Appendix Organization

- Section (A) contains technical details of BooVAE algorithm. In A.1 we provide skipped details related to the algorithm derivation: ELBO decomposition, approximated optimal prior, properties of the optimization problem and functional gradient of the objective. Next, we provide derivations for trainable flow-based prior (A.2). Finally, we discuss step-back for selection of the number of components for each task (A.3).
- Section (B) contains broader details and results for experiments in continual framework.
  - In Sections (B.1) - (B.3) we provide more detailed overview of models performance. In Section (B.1) we report NLL and diversity metric after each additional task is trained. We discuss how the performance changes on the whole test dataset as we keep training in continual learning setting. Moreover, we report and discuss these metrics for each task separately in Section (B.2). Finally, we visually study the samples from the model on each step in Section (B.3).
  - In Section (B.4) we provide additional comparison with random coresets, showing that BooVAE requires much less components to get comparable results in term of NLL and diversity metrics.
  - The implementation details for experiments are in Sec. (B.5), including architecture of neural networks and optimization details. The source code is available at [DELETED LINK, PROVIDE AS .ZIP].

A Details of the BooVAE algorithm derivations

A.1 Derivations for the optimal prior in continual framework

ELBO decomposition derivation We begin with derivations of the used decomposition in Eq.(3). We start form the ELBO definition (1) and conclude with the desirable result with several re-arrangements.

$$\mathcal{L}(\theta,\phi,\lambda) \triangleq \mathbb{E}_{\substack{p_e(\boldsymbol{x})\\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} = \mathbb{E}_{\substack{p_e(\boldsymbol{x})\\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) + \mathbb{E}_{\substack{p_e(\boldsymbol{x})\\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log \frac{\pi(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \frac{\hat{q}(\boldsymbol{z})}{\hat{q}(\boldsymbol{z})} =$$
$$= \mathbb{E}_{\substack{p_e(\boldsymbol{x})\\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) - \Big(\mathbb{E}_{p_e(\boldsymbol{x})}\mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})|\hat{q}(\boldsymbol{z})] + \underbrace{\mathbb{E}_{\substack{p_e(\boldsymbol{x})\\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log \frac{\hat{q}(\boldsymbol{z})}{\pi(\boldsymbol{z})}}_{①}\Big). \tag{18}$$

This decomposition holds for any choice of the density $\hat{q}(\boldsymbol{z})$ (under mild conditions). We make the specific choice $\hat{q}(\boldsymbol{z}) = \mathbb{E}_{p_e(\boldsymbol{x})} q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and proceed with the term ①, with celebrated Fubini's theorem:

$$① = \int d\boldsymbol{x} d\boldsymbol{z}\; p_e(\boldsymbol{x}) q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{\hat{q}(\boldsymbol{z})}{\pi(\boldsymbol{z})} = \int d\boldsymbol{z}\; \log \frac{\hat{q}(\boldsymbol{z})}{\pi(\boldsymbol{z})} \underbrace{\int d\boldsymbol{x}\; p_e(\boldsymbol{x}) q_\phi(\boldsymbol{z}|\boldsymbol{x})}_{=\mathbb{E}_{p_e(\boldsymbol{x})} q_\phi(\boldsymbol{z}|\boldsymbol{x})} =$$
$$= \int d\boldsymbol{z}\; \hat{q}(\boldsymbol{z}) \log \frac{\hat{q}(\boldsymbol{z})}{\pi(\boldsymbol{z})} = \mathrm{KL}[\hat{q}(\boldsymbol{z})|\pi(\boldsymbol{z})]. \tag{19}$$

We substitute (19) in (18) and obtain the decomposition (3):

$$\mathbb{E}_{\substack{p_e(\boldsymbol{x})\\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} = \mathbb{E}_{p_e(\boldsymbol{x})}[\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) - (\mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})|\hat{q}(\boldsymbol{z})] + \mathrm{KL}[\hat{q}(\boldsymbol{z})|\pi(\boldsymbol{z})])]. \tag{20}$$

Next we proceed to the case of $p_e(\boldsymbol{x}) = \alpha\, p_e^1(\boldsymbol{x}) + (1-\alpha)\, p_e^2(\boldsymbol{x}), \alpha \in (0;1)$. We start with a noticing:

$$\mathbb{E}_{\substack{p_e(\boldsymbol{x}) \\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log \frac{\pi(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \frac{\hat{q}(\boldsymbol{z})}{\hat{q}(\boldsymbol{z})} = \alpha\, \mathbb{E}_{\substack{p_e^1(\boldsymbol{x}) \\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log \frac{\pi(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \frac{\hat{q}_1(\boldsymbol{z})}{\hat{q}_1(\boldsymbol{z})} + (1-\alpha)\, \mathbb{E}_{\substack{p_e^2(\boldsymbol{x}) \\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log \frac{\pi(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \frac{\hat{q}_2(\boldsymbol{z})}{\hat{q}_2(\boldsymbol{z})}. \tag{21}$$

We select $\hat{q}_i(\boldsymbol{z}) = \mathbb{E}_{p_e^i(\boldsymbol{x})} q_\phi(\boldsymbol{z}|\boldsymbol{x}),\ i=1,2$ and with direct using derivations above conclude with the desirable decomposition (5):

$$L(\theta, \phi, \lambda) = \mathbb{E}_{\substack{p_e(\boldsymbol{x}) \\ q_\phi(\boldsymbol{z}|\boldsymbol{x})}} \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) - \sum_{i=1,2} \alpha_i\, (\mathbb{E}_{p_e^i(\boldsymbol{x})} \mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x}) | \hat{q}_i(\boldsymbol{z})] + \mathrm{KL}[\hat{q}_i(\boldsymbol{z}) | \pi(\boldsymbol{z})]). \tag{22}$$

Proof of the approximated optimal prior in (7) We consider the modified optimization problem over the probability density space $\mathcal{P}$ (6) with the approximation of the aggregated variational posterior of the first task $\hat{q}_1^a(\boldsymbol{z}) \approx \hat{q}_1(\boldsymbol{z})$:

$$\min_{\pi(\boldsymbol{z}) \in \mathcal{P}} \alpha\, \mathrm{KL}[\hat{q}^a(\boldsymbol{z}) | \pi(\boldsymbol{z})] + (1-\alpha)\, \mathrm{KL}[\hat{q}_2(\boldsymbol{z}) | \pi(\boldsymbol{z})]. \tag{23}$$

The optimization problem (23) is convex over $\pi(\boldsymbol{z})$ as the sum of convex functions, hence we proceed with FOC of the corresponded Lagrangian with normalization conditions:

$$\begin{aligned} &\frac{\delta}{\delta \pi}\, \alpha\, \mathrm{KL}[\hat{q}^a(\boldsymbol{z}) | \pi(\boldsymbol{z})] + (1-\alpha)\, \mathrm{KL}[\hat{q}_2(\boldsymbol{z}) | \pi(\boldsymbol{z})] + \lambda \left(\int d\boldsymbol{z}\, \pi(\boldsymbol{z}) - 1 \right) = \\ &= - \left(\alpha \frac{\hat{q}^a(\boldsymbol{z})}{\pi(\boldsymbol{z})} + (1-\alpha) \frac{\hat{q}_2(\boldsymbol{z})}{\pi(\boldsymbol{z})} \right) + \lambda = 0. \end{aligned} \tag{24}$$

By rearranging terms and normalize the solution, we conclude with the stated result:

$$\pi^{1,2}(\boldsymbol{z}) = \alpha \hat{q}_1^a(\boldsymbol{z}) + (1-\alpha) \hat{q}_2(\boldsymbol{z}). \tag{25}$$

Proof of the bi-convexity of the optimization problem (8) over h and β We consider the functional:

$$\alpha\, \mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})] + (1-\alpha)\, \mathrm{KL}[\hat{q}_2(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})]). \tag{26}$$

We show bi-convexity over h and β for the term $\mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})]$ as other is of the same form and sum of convex functions is a convex function. The convexity over h follows from convexity of KL divergence:

$$\begin{aligned} &\mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta(\alpha h_1(\boldsymbol{z}) + (1-\alpha) h_2(\boldsymbol{z})] \leq \\ &\leq (1-\beta)\, \mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | \pi^1(\boldsymbol{z})] + \beta\, \left(\alpha\, \mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | h_1(\boldsymbol{z})] + (1-\alpha)\, \mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | h_2(\boldsymbol{z})] \right). \end{aligned} \tag{27}$$

We check convexity over β by expecting of the second derivative:

$$\begin{aligned} &\nabla_\beta \mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})] = - \int d\boldsymbol{z}\, \hat{q}_1^a(\boldsymbol{z}) \frac{h(\boldsymbol{z}) - \pi^1(\boldsymbol{z})}{(1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})}, \\ &\nabla_\beta^2 \mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})] = \int d\boldsymbol{z}\, \hat{q}_1^a(\boldsymbol{z}) \left(\frac{h(\boldsymbol{z}) - \pi^1(\boldsymbol{z})}{(1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})} \right)^2 > 0. \end{aligned} \tag{28}$$

Derivation of the functional gradient for the optimization problem (8) over h The functional of our interest is $\alpha\, \mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})] + (1-\alpha)\, \mathrm{KL}[\hat{q}_2(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})])$. We consider the perturbation of the argument $\pi^1(\boldsymbol{z})$ with $h(\boldsymbol{z})$ as following $(1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})$. We start from the linearization of the first term:

$$\begin{aligned} &\mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | (1-\beta)\pi^1(\boldsymbol{z}) + \beta h(\boldsymbol{z})] = - \int d\boldsymbol{z}\, \hat{q}_1^a(\boldsymbol{z}) \left\{ \log \frac{\pi^1(\boldsymbol{z})}{q_1^a(\boldsymbol{z})} + \log \left[1 + \beta \left(\frac{h(\boldsymbol{z})}{\pi^1(\boldsymbol{z})} - 1 \right) \right] \right\} = \\ &= \{^{\log(1+x) = x + o(x),}_{x \to 0}\} = \mathrm{KL}[\hat{q}_1^a(\boldsymbol{z}) | \pi^1(\boldsymbol{z})] - \beta \left(\int d\boldsymbol{z}\, h(\boldsymbol{z}) \frac{\hat{q}_1^a(\boldsymbol{z})}{\pi^1(\boldsymbol{z})} - 1 \right) + o(\beta). \end{aligned} \tag{29}$$

With application of this result to the second term, we obtain:

$$\alpha\,\mathrm{KL}[\hat{q}_1^a(\boldsymbol{z})|(1-\beta)\pi^1(\boldsymbol{z})+\beta h(\boldsymbol{z})]+(1-\alpha)\,\mathrm{KL}[\hat{q}_2(\boldsymbol{z})|(1-\beta)\pi^1(\boldsymbol{z})+\beta h(\boldsymbol{z})])=$$
$$\alpha\,\mathrm{KL}[\hat{q}_1^a(\boldsymbol{z})|\pi^1(\boldsymbol{z})]+(1-\alpha)\,\mathrm{KL}[\hat{q}_2(\boldsymbol{z})|\pi^1(\boldsymbol{z})]-\beta\underbrace{\left(\int d\boldsymbol{z}\;h(\boldsymbol{z})\left[\alpha\frac{\hat{q}_1^a(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}+(1-\alpha)\frac{\hat{q}_2(\boldsymbol{z})}{\pi^1(\boldsymbol{z})}\right]-1\right)}_{①}+o(\beta). \tag{30}$$

The term ① is the functional gradient. We project direction h to match it at the optimization problem (12).

A.2 BooVAE for VAE with flow-based prior

Consider the ELBO objective:

$$\mathcal{L}(\theta,\phi)\triangleq\mathbb{E}_{p_e(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z})-\log q_\phi(\boldsymbol{z}|\boldsymbol{x})\right). \tag{31}$$

The simplest choice of the prior $p(\boldsymbol{z})$ is the standard normal distribution. Instead, in order to improve ELBO, one could obtain multi-modal prior distribution $p(\boldsymbol{z})$ by using learnable bijective transformation f:

$$\boldsymbol{v}\sim p_0(\boldsymbol{v}),\,\boldsymbol{z}=f(\boldsymbol{v}). \tag{32}$$

This induce the following density over the prior in $\boldsymbol{z}$-space:

$$\pi(\boldsymbol{z})=\int p(\boldsymbol{v})\delta_{\boldsymbol{z}}(f(\boldsymbol{v}))\;d\boldsymbol{v}=p_0(f^{-1}(\boldsymbol{z}))|J_{\boldsymbol{z}}^{f^{-1}}|. \tag{33}$$

We could obtain optimal prior in z-space by using the following decomposition of the ELBO:

$$\mathbb{E}_{p_e(\boldsymbol{x})}\left[\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\log p_\theta(\boldsymbol{x}\|\boldsymbol{z})-\mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})\|\hat{q}(\boldsymbol{z})]-\mathrm{KL}[\hat{q}(\boldsymbol{z})\|\pi(\boldsymbol{z})]\right], \tag{34}$$

where $\hat{q}(\boldsymbol{z})=\mathbb{E}_{p_e(\boldsymbol{x})}q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is the aggregated variational posterior. As KL-divergence is non-negative, the global maximum over π is reached when: $\pi(\boldsymbol{z})=\mathbb{E}_{p_e(\boldsymbol{x})}q_\phi(\boldsymbol{z}|\boldsymbol{x})$. In oder to reach it, we should match:

$$\pi(\boldsymbol{z})=\hat{q}(\boldsymbol{z})\Longrightarrow p_0(f^{-1}(\boldsymbol{z}))|J_{\boldsymbol{z}}^{f^{-1}}|=\mathbb{E}_{p_e(\boldsymbol{x})}q_\phi(\boldsymbol{z}|\boldsymbol{x}). \tag{35}$$

It could be done with tuning base distribution p_0 and parameters of the transformation f. In order to use BooVAE algorithm, we decouple this updates. The parameters of the transformation f are updated in the Maximization step, together with encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ and base prior distribution $p_0(\cdot)$ is fixed. So this step is the same as optimization step in VAE training. In the Minorization step, we need to update $p_0(\cdot)$. In order to do this, we came back to the $\boldsymbol{v}$-space.

$$\begin{aligned}\mathrm{KL}[\pi(\boldsymbol{z})|\hat{q}(\boldsymbol{z})]&=\int\pi(\boldsymbol{z})\log\frac{\pi(\boldsymbol{z})}{\hat{q}(\boldsymbol{z})}\;d\boldsymbol{z}=\left\{\begin{smallmatrix}\boldsymbol{z}=f(\boldsymbol{v}),\\ d\boldsymbol{z}=|J_{\boldsymbol{v}}^f|d\boldsymbol{v}\end{smallmatrix}\right\}=\\ &=\int p_0(\boldsymbol{v})|J_{f(\boldsymbol{v})}^{f^{-1}}|\log\frac{p_0(f^{-1}(f(\boldsymbol{v})))|J_{f(\boldsymbol{v})}^{f^{-1}}|}{\hat{q}(f(\boldsymbol{v}))}\;|J_{\boldsymbol{v}}^f|d\boldsymbol{v}=\\ &\int p_0(\boldsymbol{v})\log\frac{p_0(\boldsymbol{v})}{\hat{q}(f(\boldsymbol{v})|J_{\boldsymbol{v}}^f|}\;d\boldsymbol{v}=\mathrm{KL}[p_0(\boldsymbol{v})|\hat{q}(f(\boldsymbol{v})|J_{\boldsymbol{v}}^f|].\end{aligned} \tag{36}$$

We conclude with the same problem of matching aggregation posterior, but in the $\boldsymbol{v}$-space.

A.3 Step-back for components

On practice, it is not obvious, what is the optimal number of components in the prior. We have experimentally observed, that excessive amount of the prior component can be as harmful as the insufficient number of components. This happens because we learn each component as a pseudoinput to the encoder. Throughout VAE training we update parameters of the encoder by maximizing the ELBO, meaning that components are also unintentionally updated and some of them become irrelevant.

To circumvent this disadvantage without the need to retrain VAE with different number components, we suggest to prune the prior after the maximal number of components is reached. This approach

allows us in theory to select this maximal number to be as large as possible and then remove all the relevant ones during pruning.

Pruning is performed as minimization of the KL-divergence between optimal prior and current approximation with the respect to the weights if the mixture. This optimization procedure is performed in the latent space and only ones during training. Therefore, it almost does not influence the training time. In the experiment setting we report the **maximal** number of components. That is total number of prior components before pruning.

B Details of the Experiments and Ablation Study

All the experiments we performed on a single NVIDIA Tesla V100 GPU.

B.1 Results in continual learning setting

In Tables (2), (3) we provide full results for continual learning experiments, which are illustrated by Fig.(2),(3). We provide negative log-likelihood in Table (2) and KL-divergence used as diversity measure in Table (3). The first column in both tables states how many tasks did the VAE see in total and the value in the table indicates value of negative log-likelihood or diversity metrics on the test set containing current and all the previous tasks.

We add comparison with multi-head architectures. We observe that BooVAE is capable to achieve results comparable to multi-head architecture. We find this to be a good result, because fixed architecture that we use has approximately 6 times less parameters. Moreover, when computing NLL on the test set multi-head architecture requires task tables in order to use proper head, while in case of BooVAE the test evaluation is performed in unsupervised manner. Even though NLL on a test set is a default way of assessing VAEs, during our experiments, we've observed that good NLL scores do not always correspond to diverse samples from the prior in the continual setting. See samples in Fig.(12),(13),(14) for the confirmation of this observation. For example, random coresets are performing well in terms of NLL, but we can see that they are still prone to catastrophic forgetting when it comes to generating samples. Therefore, we were interested in quantitative evaluation of the samples, produced by the VAE on each step. As the diversity score we calculate KL-divergence between desired $P(x)$ and observed $Q(x)$ distribution of generated images over the classes. Desired distribution is multinomial with equal probabilities for each class. To evaluate observed distribution of classes we generate $N = 10^3$ samples from VAE and classify them using neural network, trained on the same dataset as the VAE. Then we calculate the proportion of images from the class i in the generated sample and evalutate KL-divergence:

$$P(x=i)=p_i=\frac{1}{T},\ i\in\{1,\dots,T\}, Q(x=i)=\hat{p}_i=\frac{N_i}{N},\ i\in\{1,\dots,T\},\ \mathrm{KL}[P(x)|Q(x)]=\sum_{i=1}^{T}\frac{1}{T}\log\frac{\frac{1}{T}}{\hat{p}_i}. \tag{37}$$

If the model generates images from all the classes in equal proportions, the value of the metrics is zero. The large is KL-divergence, the less diversity is there in the samples. We would like to note, that this metric is reflecting the situation that we observe in Figures 12, 13, 14 and confirms that BooVAE is dealing with catastrophic forgetting much better that other methods. Moreover, we observe that even though the combination of BooVAE with VCL is the best in terms of reconstruction error (NLL), it produce more blurred samples and therefore the classification network make more error with is reflected on the KL-divergence.

B.2 Metrics for each task separately

In Fig.(9),(10) we report test NLL and diversity metrics from the previous section for each task separately. Each subgraph shows performance of the VAE on one specific task (e.g. on '0's, '1's, etc. for MNIST), depending on the total number of tasks seen by the model in continual setting. Each line begins when the class appears in the train set for the first time. We observe, how the performance on this task changes as wee keep updating the model on the new tasks. We expect the line to be as flat as possible. This means, that quality of reconstructions and proportion of samples does not get worse when new tasks arrive. In Fig.(10) we show each term in the KL-divergence as a characteristic of

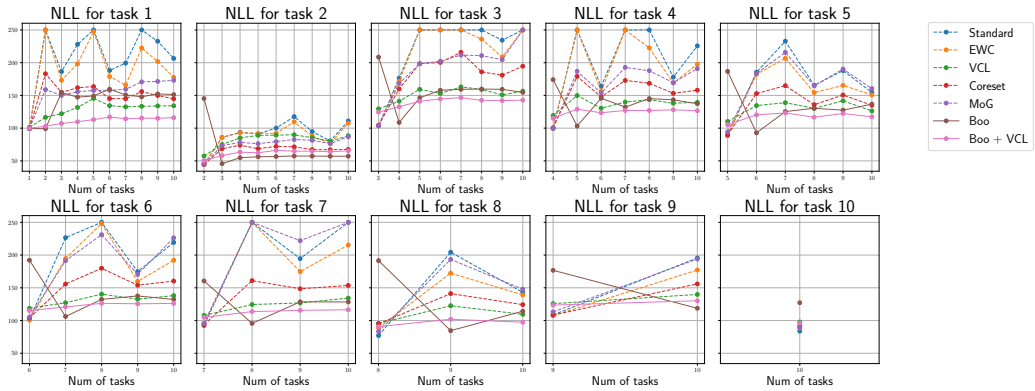
Table 2: NLL on a test set, averaged over 5 runs with standard deviation in the brackets. Each model was trained on 10 tasks in a continual setting, in multi-head models new encoder and extra decoder layer was added for every new task. We use ***bold italics*** to denote the best result among all the models and **bold** to denote best among the models with one encoder-decoder pair for all the tasks.

# Tasks	Standard	EWC	VCL	Coreset	MoG	Boo	Boo + VCL	Multihead	Multihead + EWC
					MNIST				
2	343.5 (26)	258.3 (12)	84.8 (0.4)	108.6 (2.2)	98.1 (3.2)	96.1 (0.9)	***74.9 (0.3)***	399.2 (6.8)	119.6 (18.5)
3	122.0 (2.3)	118.9 (1.5)	107.6 (0.6)	106.9 (0.2)	106.9 (2.7)	101.3 (0.4)	***95.2 (0.7)***	203.9 (1.5)	114.2 (6.9)
4	146.1 (0.3)	138.5 (1.5)	118.3 (0.5)	121.8 (1.3)	123.7 (0.8)	111.9 (0.1)	***104.1 (0.9)***	217.6 (3.5)	115.3 (7.0)
5	197.0 (5.7)	187.4 (2.2)	129.9 (1.2)	138.1 (1.9)	141.2 (3.5)	121.0 (2.3)	***109.1 (0.8)***	281.5 (2.6)	115.9 (8.0)
6	164.3 (3.8)	158.8 (2.8)	126.1 (0.8)	135.8 (1.1)	144.5 (3.4)	120.8 (0.2)	***113.5 (0.9)***	215.0 (2.3)	***113.5 (5.8)***
7	205.2 (5.6)	184.2 (3.8)	128.0 (0.8)	144.2 (1.7)	161.9 (5.8)	122.9 (0.3)	***113.7 (0.8)***	247.6 (4.2)	***113.7 (6.1)***
8	213.2 (9.2)	187.0 (2.5)	125.6 (0.6)	141.4 (2.5)	168.7 (7.8)	122.4 (0.8)	***111.2 (0.4)***	301.0 (5.8)	***112.2 (6.6)***
9	171.0 (3.6)	157.7 (3.7)	127.4 (0.5)	137.5 (2.8)	163.3 (2.4)	124.5 (2.7)	***114.5 (0.9)***	210.6 (5.0)	***112.3 (5.9)***
10	186.8 (2.3)	169.3 (3.2)	125.7 (0.6)	137.0 (3.5)	180.1 (1.3)	124.5 (1.9)	***112.7 (1.1)***	256.5 (6.3)	***111.3 (5.6)***
					notMNIST				
2	329.5 (24)	298.9 (9.0)	221.6 (3.1)	241.8 (2.4)	251.1 (4.7)	238.6 (0.7)	**210.4 (0.9)**	497.9 (21)	***188.1 (1.7)***
3	412.3 (23)	346.5 (7.1)	234.1 (5.5)	243.5 (4.6)	288.5 (11)	245.6 (3.1)	**206.3 (1.7)**	702.6 (58)	***179.8 (1.8)***
4	299.2 (6.2)	290.3 (8.2)	253.4 (5.4)	227.5 (1.4)	271.2 (6.3)	238.8 (3.6)	**215.7 (2.3)**	662.5 (38)	***175.2 (3.5)***
5	321.4 (9.6)	290.1 (4.7)	238.5 (3.8)	235.0 (4.9)	274.2 (4.6)	251.1 (7.1)	**212.7 (1.1)**	644.0 (33)	***174.2 (2.3)***
6	329.1 (15)	316.8 (11)	248.3 (4.3)	228.1 (3.5)	301.7 (18)	247.5 (11)	**214.6 (0.9)**	803.7 (72)	***170.2 (2.0)***
7	310.9 (20)	294.3 (3.6)	241.3 (3.3)	231.2 (3.1)	288.8 (7.1)	255.8 (5.9)	**216.4 (0.9)**	724.6 (39)	***170.8 (2.1)***
8	338.6 (35)	316.4 (14)	254.5 (7.1)	230.6 (2.0)	320.4 (15)	263.1 (4.2)	**224.2 (1.6)**	757.9 (47)	***169.1 (2.0)***
9	339.1 (12)	314.1 (3.8)	234.2 (3.1)	229.0 (2.6)	317.8 (13)	252.7 (5.3)	**214.8 (1.3)**	905.3 (56)	***161.5 (1.6)***
10	402.4 (13)	374.4 (12)	242.2 (3.3)	233.7 (2.3)	358.6 (14)	261.4 (8.7)	**214.3 (1.0)**	951.2 (47)	***158.7 (2.2)***
					fashionMNIST				
2	259.1 (4.9)	270.9 (9.7)	223.5 (0.6)	230.3 (1.3)	256.4 (13)	224.5 (0.6)	**218.8 (0.6)**	383.3 (15)	***231.0 (11.4)***
3	290.9 (3.8)	284.1 (6.8)	253.8 (0.4)	257.8 (0.6)	266.1 (2.7)	252.9 (0.6)	***249.7 (0.4)***	327.1 (9.6)	***250.8 (4.2)***
4	273.5 (2.4)	269.5 (1.8)	250.9 (0.6)	254.1 (1.5)	273.4 (3.8)	244.7 (0.4)	***242.1 (0.6)***	336.1 (9.5)	***240.8 (4.0)***
5	272.0 (1.8)	268.1 (1.0)	255.9 (0.4)	255.2 (0.9)	266.3 (3.2)	250.8 (0.8)	***248.6 (0.2)***	333.6 (3.7)	***247.7 (3.5)***
6	507.8 (46)	447.4 (20)	262.7 (2.4)	258.1 (0.5)	329.3 (14)	245.2 (3.2)	***243.5 (1.7)***	675.8 (14.1)	***242.8 (5.6)***
7	276.5 (3.2)	271.0 (2.8)	257.2 (0.5)	255.7 (0.7)	266.3 (4.7)	250.0 (1.1)	***248.5 (0.3)***	382.7 (9.4)	***247.7 (3.2)***
8	1468 (585)	540.8 (25)	254.7 (1.6)	258.0 (0.3)	314.8 (18)	243.5 (1.4)	***240.5 (0.8)***	606.2 (35.7)	***239.7 (3.0)***
9	310.1 (19)	284.1 (1.4)	261.9 (0.9)	260.8 (0.8)	285.1 (3.0)	249.3 (1.1)	***249.1 (1.1)***	502.7 (30.3)	***248.9 (4.5)***
10	799.9 (273)	399.2 (27)	268.6 (1.5)	263.0 (1.6)	330.9 (8.4)	250.6 (4.6)	***248.7 (0.5)***	569.1 (28.4)	***247.3 (4.7)***

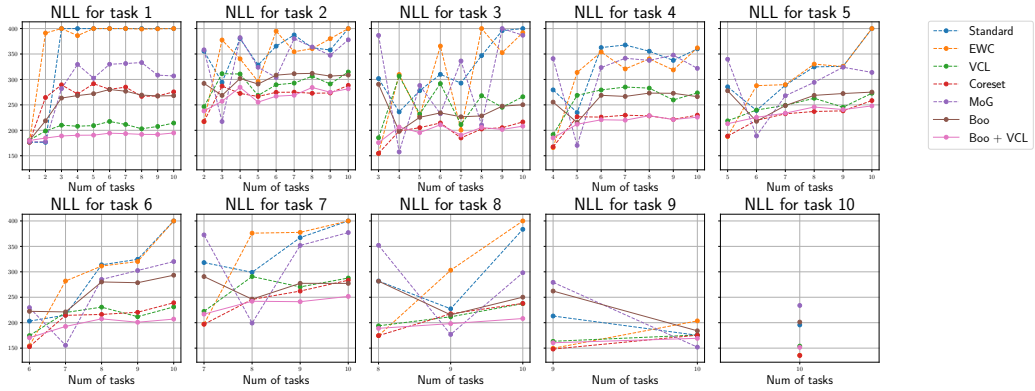
the per task diversity, which is equal to the $\frac{1}{K}\left(\log\frac{1}{K}-\log\hat{p}_i\right)$, $i\in\{1,\cdots,K\}$. Ideally, this value should be 0 everywhere. If it is positive, then there is not enough images from a given class in the sample. And vice versa, in case of negative value, there are too many samples from a given class. As we can see from the plots, BooVAE is extremely close to the desired behaviour. If the method suffer from catastrophic forgetting, it produces too many samples when the task is seen for the first time (green zone on the graph) and not enough samples later on (red area), since it forgets how to produce this samples.

Table 3: Diversity results, averaged over 5 runs with standard deviation in the brackets. The lower is better, we use **bold** to denote the best model with one encoder-decoder pair.

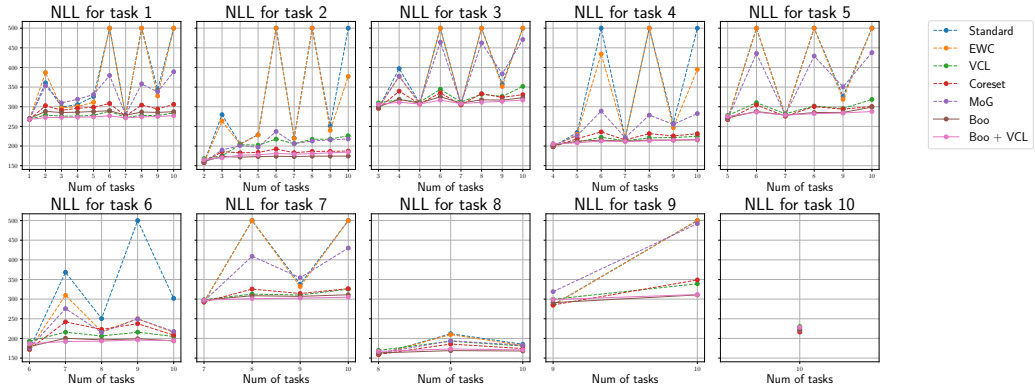
# Tasks	Standard	EWC	VCL	Coreset	MoG	Boo	Boo + VCL
				MNIST			
2	0.69 (0.00)	0.69 (0.00)	0.10 (0.02)	0.69 (0.00)	**0.00 (0.00)**	**0.00 (0.00)**	0.21 (0.03)
3	1.04 (0.00)	1.01 (0.01)	0.23 (0.02)	1.00 (0.01)	**0.00 (0.00)**	**0.00 (0.00)**	0.29 (0.02)
4	1.33 (0.01)	1.32 (0.02)	0.24 (0.02)	1.29 (0.02)	**0.00 (0.00)**	**0.00 (0.00)**	0.34 (0.02)
5	1.57 (0.01)	1.58 (0.01)	0.12 (0.02)	1.54 (0.01)	0.24 (0.00)	**0.00 (0.00)**	0.30 (0.02)
6	1.68 (0.03)	1.67 (0.02)	0.57 (0.04)	1.58 (0.03)	0.51 (0.01)	**0.00 (0.00)**	0.64 (0.01)
7	1.92 (0.01)	1.88 (0.02)	0.65 (0.04)	1.82 (0.01)	0.67 (0.14)	**0.00 (0.00)**	0.58 (0.02)
8	2.02 (0.02)	2.01 (0.01)	0.63 (0.04)	1.91 (0.04)	0.88 (0.15)	**0.03 (0.05)**	0.59 (0.02)
9	2.09 (0.02)	2.07 (0.02)	0.67 (0.02)	1.90 (0.04)	1.07 (0.14)	**0.08 (0.11)**	0.63 (0.02)
10	2.26 (0.01)	2.22 (0.01)	0.78 (0.02)	2.09 (0.01)	1.23 (0.14)	**0.11 (0.15)**	0.74 (0.01)
				notMNIST			
2	0.62 (0.02)	0.62 (0.02)	**0.00 (0.00)**	0.57 (0.05)	0.22 (0.08)	0.01 (0.01)	0.10 (0.03)
3	1.04 (0.01)	1.02 (0.01)	**0.02 (0.01)**	0.93 (0.03)	0.53 (0.16)	**0.01 (0.00)**	0.31 (0.05)
4	1.11 (0.09)	1.12 (0.03)	0.06 (0.01)	0.92 (0.05)	0.72 (0.18)	**0.02 (0.02)**	0.51 (0.07)
5	1.35 (0.04)	1.32 (0.03)	0.04 (0.01)	0.94 (0.04)	0.91 (0.19)	**0.01 (0.01)**	0.44 (0.06)
6	1.49 (0.08)	1.52 (0.03)	0.10 (0.02)	1.07 (0.06)	1.01 (0.19)	**0.05 (0.03)**	0.40 (0.06)
7	1.52 (0.12)	1.51 (0.04)	0.14 (0.03)	1.00 (0.05)	1.22 (0.24)	**0.05 (0.04)**	0.40 (0.06)
8	1.46 (0.14)	1.53 (0.06)	0.35 (0.07)	0.76 (0.03)	1.40 (0.21)	**0.07 (0.03)**	0.44 (0.07)
9	1.10 (0.39)	1.09 (0.09)	0.23 (0.03)	0.37 (0.08)	1.49 (0.22)	**0.14 (0.10)**	0.62 (0.06)
10	1.95 (0.06)	1.94 (0.05)	0.35 (0.06)	1.22 (0.03)	1.59 (0.21)	**0.20 (0.15)**	0.60 (0.05)
				fashionMNIST			
2	0.68 (0.00)	0.67 (0.01)	0.01 (0.00)	0.65 (0.01)	0.03 (0.03)	**0.00 (0.00)**	**0.00 (0.00)**
3	1.02 (0.00)	1.00 (0.02)	0.21 (0.02)	0.89 (0.03)	0.19 (0.07)	**0.00 (0.00)**	**0.00 (0.00)**
4	1.28 (0.01)	1.26 (0.02)	0.55 (0.01)	1.15 (0.03)	0.34 (0.11)	**0.00 (0.00)**	**0.00 (0.00)**
5	1.34 (0.01)	1.30 (0.02)	0.30 (0.01)	1.10 (0.04)	0.37 (0.09)	**0.00 (0.00)**	0.01 (0.00)
6	1.78 (0.00)	1.77 (0.01)	0.31 (0.02)	1.68 (0.02)	0.53 (0.14)	**0.00 (0.00)**	0.01 (0.00)
7	1.31 (0.03)	1.31 (0.05)	0.61 (0.03)	1.11 (0.04)	0.63 (0.12)	**0.01 (0.01)**	**0.02 (0.01)**
8	2.08 (0.00)	2.08 (0.00)	0.36 (0.02)	1.96 (0.02)	0.67 (0.12)	**0.00 (0.00)**	0.03 (0.01)
9	2.07 (0.01)	2.10 (0.00)	0.86 (0.04)	1.96 (0.02)	0.77 (0.11)	**0.01 (0.01)**	**0.02 (0.01)**
10	2.24 (0.01)	2.24 (0.01)	0.63 (0.03)	2.03 (0.03)	0.85 (0.10)	**0.02 (0.02)**	0.04 (0.01)



(a) MNIST

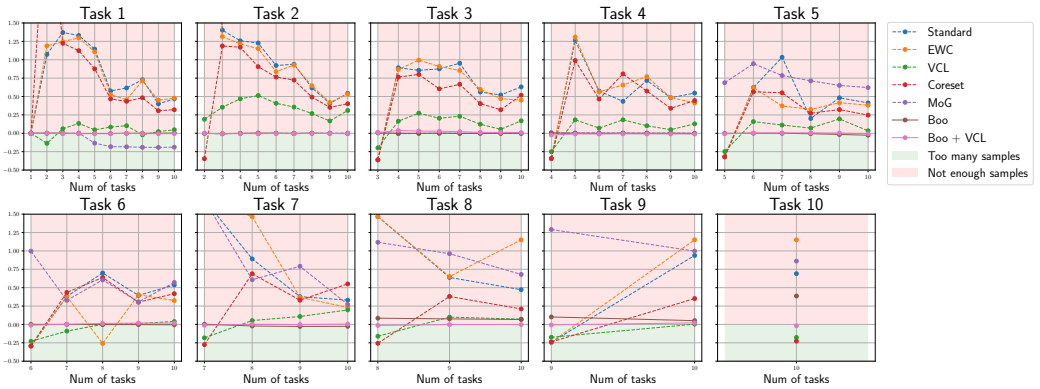


(b) notMNIST

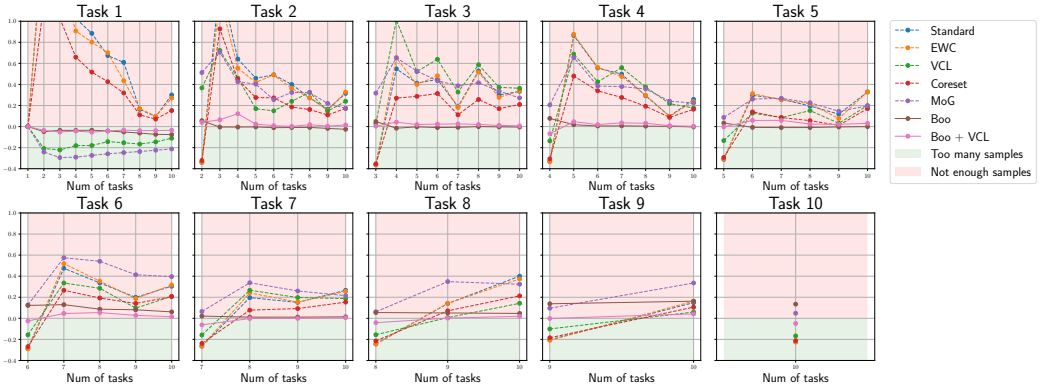


(c) Fashion MNIST

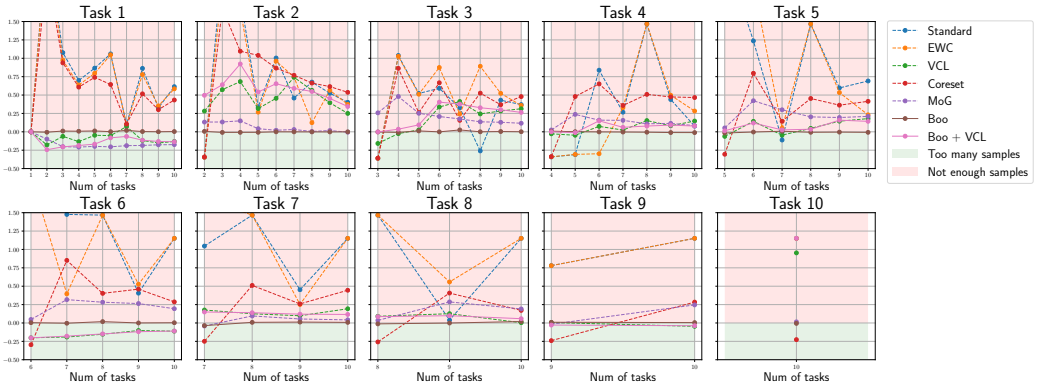
Figure 9: NLL on the test dataset for each task separately averaged over 5 runs. The lower is better.



(a) MNIST



(b) notMNIST



(c) Fashion MNIST

Figure 10: Diversity score for each computed for each class separately. Each subgraph shows values $^1/_K\left(\log {}^1/_K - \log \hat{p}_i\right)$, $i \in \{1, \cdots, K\}$ for the task k. That is one term from the sum (KL-divergence), which shows difference between ideal and generated proportion of images for a specific class. On the x axis there is total number of tasks seen by the model. Negative value means that model generates too many images of the class, positive — not enough images from the given class.

B.3 Examples of generated samples

Figures (11),(12),(13),(14) contain samples from different VAE. Each row-block corresponds to the total number of tasks seen by the model, while each columns corresponds to a different model. We can clearly see, that BooVAE generates diverse samples after training on all the tasks (last row). We also observe that addition of the regularization-based approaches makes samples from the models more blurred. We assume that this is exactly reflected in the worse diversity metrics for BooVAE with VCL. Even though the samples are still diverse, they are are sometimes of lower quality and classification network makes more mistakes.

These samples provide qualitative proof of the generation diversity metrics, shown in Figure (2) and in Tables (3),(1).

Figure 11: Samples from prior after training on different number of tasks in the continual setting: CelebA dataset. Each row-wise block from the top to the bottom corresponds to the cumulative increasing number of tasks. Each column-wise block corresponds to the particular model.

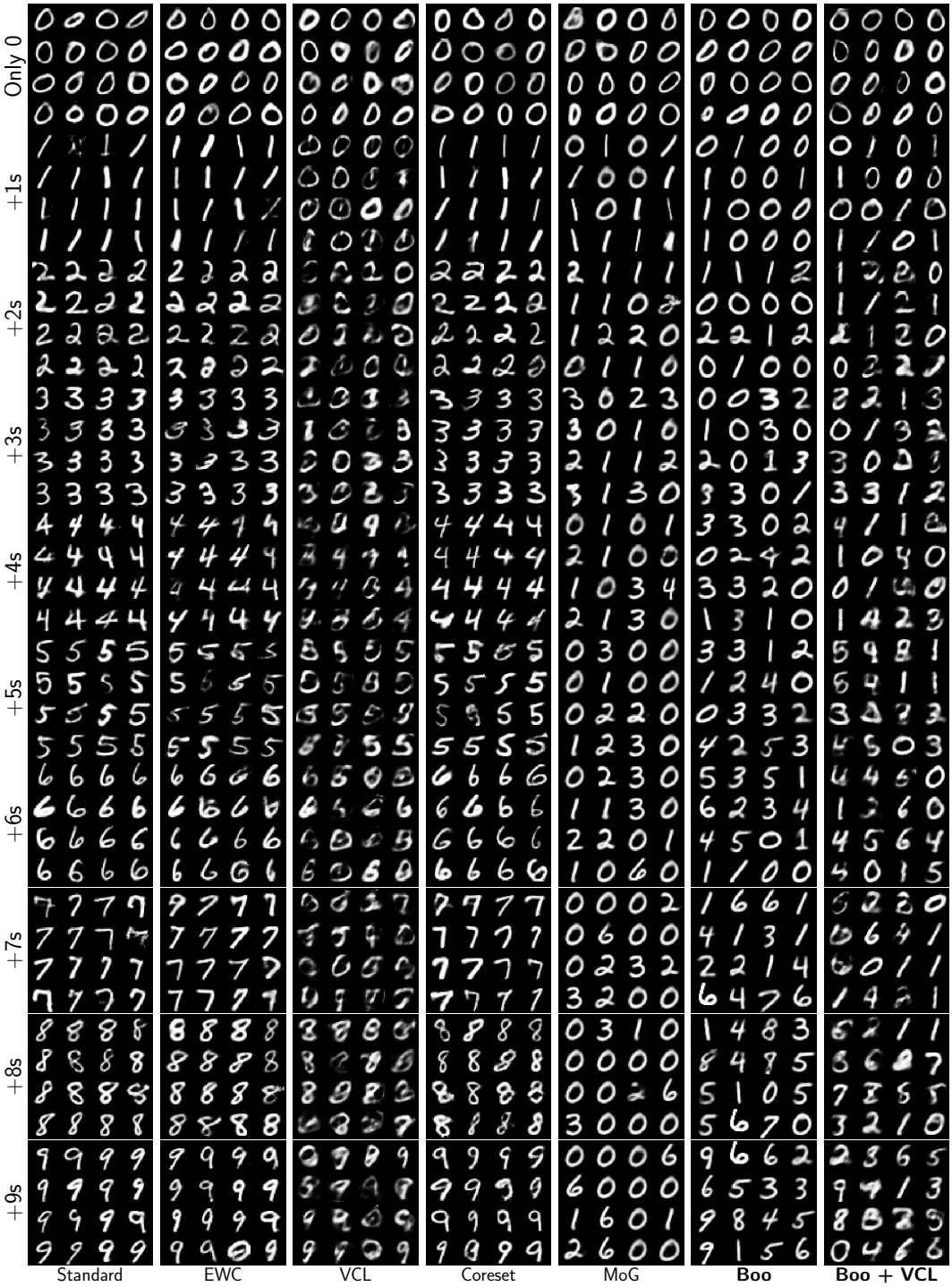


Figure 12: Samples from prior after training on different number of tasks in the continual setting: MNIST dataset. Each row-wise block from the top to the bottom corresponds to the cumulative increasing number of tasks. Each column corresponds to the particular model.

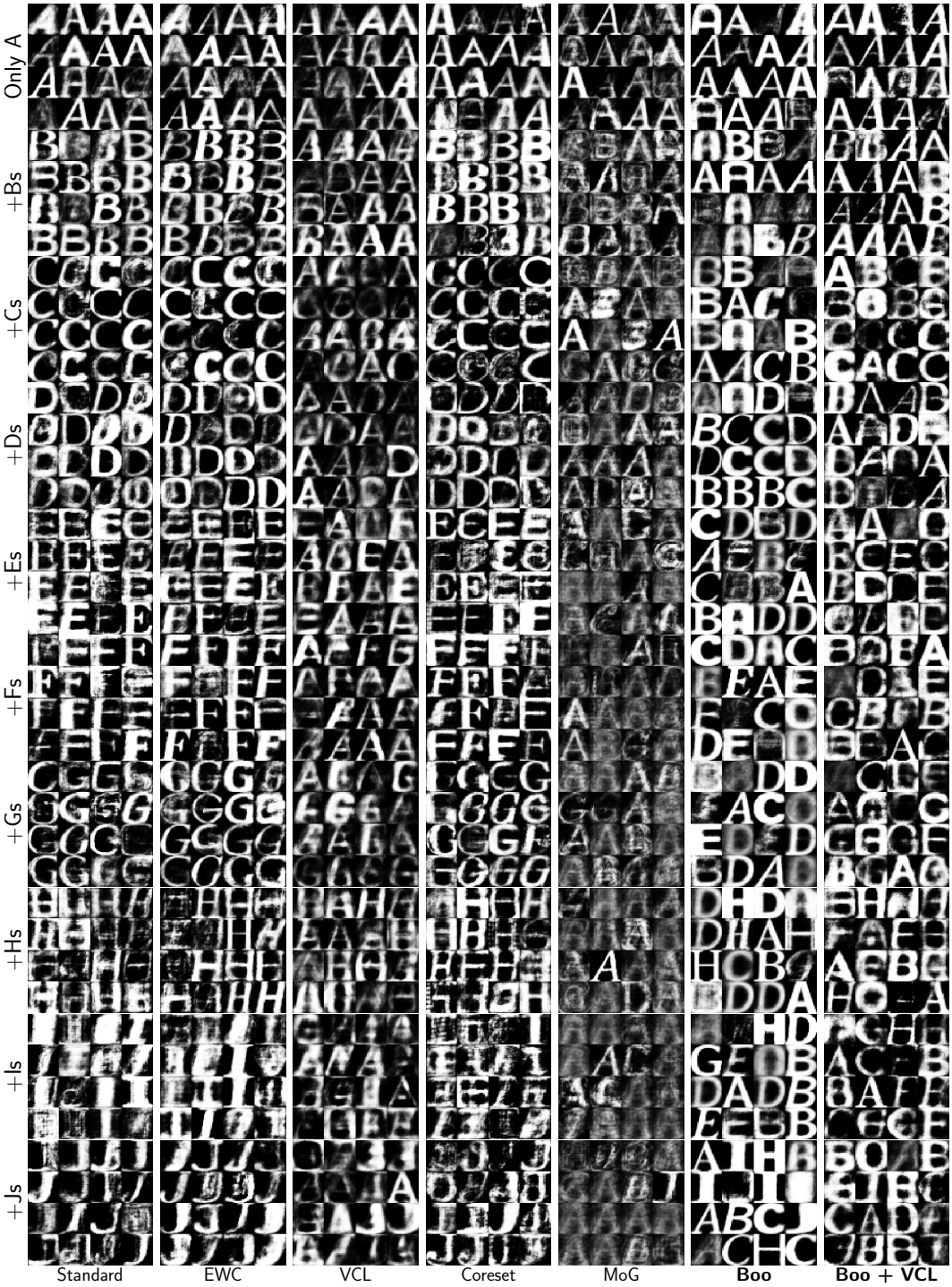


Figure 13: Samples from prior after training on different number of tasks in the continual setting: notMNIST dataset. Each row-wise block from the top to the bottom corresponds to the cumulative increasing number of tasks. Each column-wise block corresponds to the particular model.

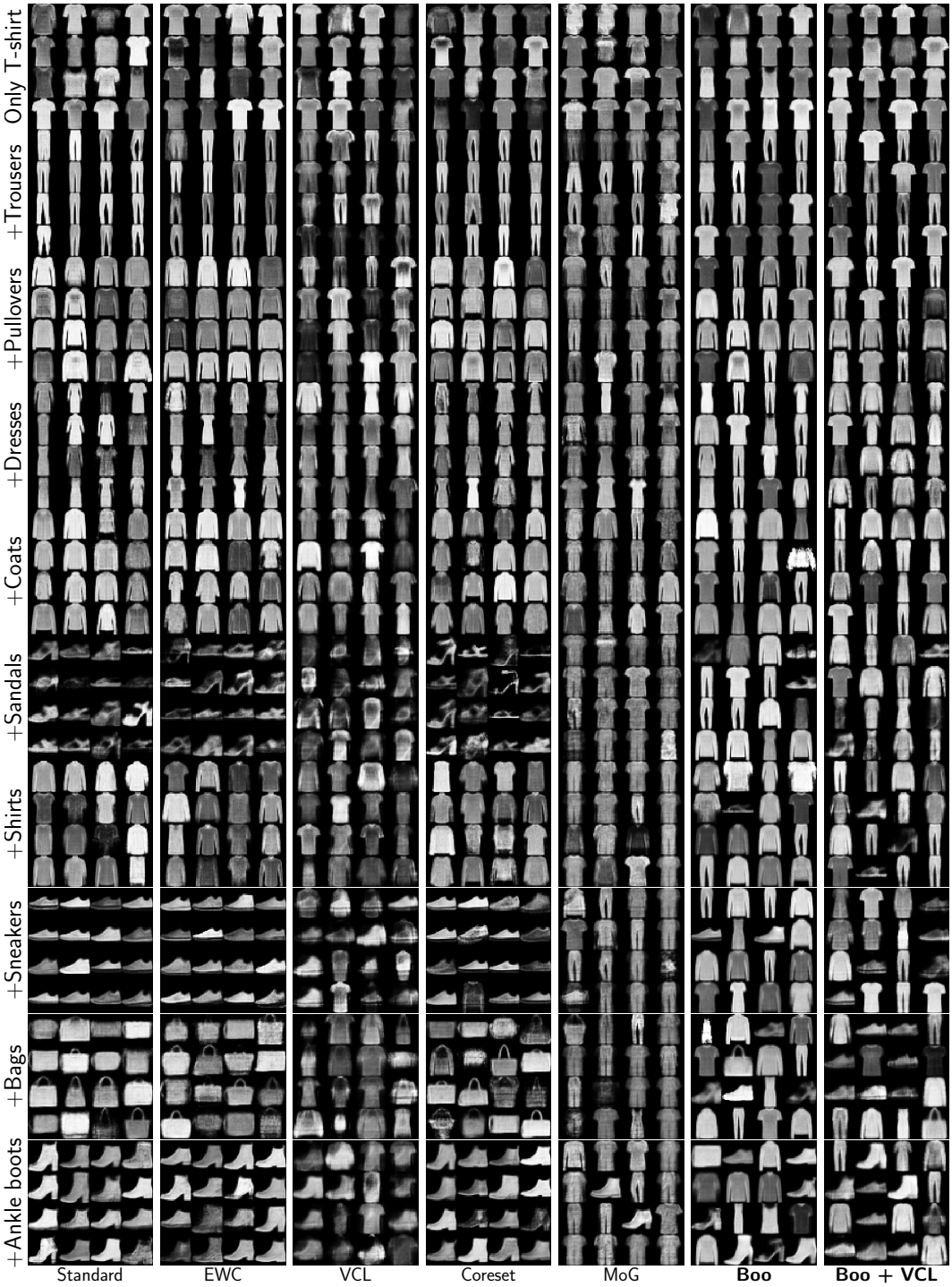


Figure 14: Samples from prior after training on different number of tasks in the continual setting: Fashion MNIST dataset. Each row-wise block from the top to the bottom corresponds to the cumulative increasing number of tasks. Each column-wise block corresponds to the particular model.

B.4 Random Coreset Size

In all the experiments we use the size of the random coreset equal to the maximal number of components in BooVAE, which is 15 for all the MNIST dataset and 40 for CelebA. Since random coreset basically means that we store a subset of the training data from the previous tasks, it is always possible to find such size of the random coreset, that there is no catastrophic forgetting at all. In this section we show, how large the random coreset should be to achieve results comparable to BooVAE with 15 components.

We observe that on MNIST dataset only random coreset of size 500 per task results in better results in terms of both NLL and KL divergence. Lower size of the random coreset does not produce samples that are as diverse as samples from BooVAE. For Fashion MNIST we observe that situation with NLL is similar, but even 500 samples is not enough to get diverse enough samples from the model.

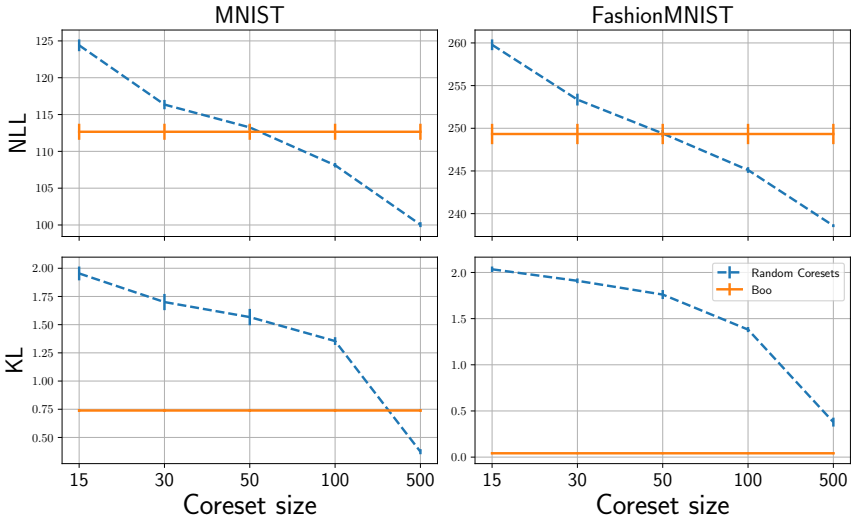


Figure 15: Negative Loglikelohood (top row) and KL, which assesses diversity of generated samples (bottom row) for MNIST and Fashion MNIST dataset for different sizes of Random Corsets after training on 10 tasks.

B.5 Architecture and Optimization details

Optimization We use validation dataset to select hyperparameters for a simple VAE (with a standard Normal prior, trained on the whole training dataset) for each dataset. After that we fix these hyperparemeters for all the methods used in the experiments in the continual learning setting.

For MNIST and FashionMNIST we randomly remove 10'000 images from the train dataset for validation. For notMNIST we use small version of the dataset (19k images in total) and remove 10% as a test data and 10% more as validation. For CelebA dataset we use split provided by the dataset authors.

We use Adam to perform the optimization for all the datasets with LR scheduler, that reduce learning rate by the `factor` when the loss is not decreasing for `patience` number of epochs. In the Table 4 we provide all the parameters of the optimization procedure. For CelebA dataset we use $\beta-$VAE with β annealing. The value of β is gradually increasing from 0 to 2 during the first 10 epochs.

MNIST architecture We've used MLP with 3 linear layers and LeakyReLU activations both for the encoder and decoder in all the cases. Detailed architectures are presented in Table 5.

CelebA architecture For CelebA dataset we've used convolutional NN. We use 2D convolutions with kernel size 5×5, Batch normalization and ReLU activations for encoder and symmetric architecture but with transposed convolutions for the decoder. See all the details in Table 6.

Table 4: Optimization parameters used in the experiments.

Parameter	MNIST	notMNIST	FashionMNIST	CelebA
Batch-size	250	250	500	512
Initial Learning rate	5e–4	5e–4	5e–4	1e–3
LR scheduler patience	30	30	30	9
LR scheduler factor	0.5	0.5	0.5	0.5
Max epochs	500	500	1000	300
Early stopping (# ep. without improvement)	50	50	50	15
Latent dimension	40	40	40	128
Beta annealing	No	No	No	From 0 to 2 during first 10 epochs
# components per task (BooVAE)	15	15	15	40
Regularization weight (BooVAE, eq. (17))	1	1	1	1

Table 5: MLP architectures for MNIST, notMNIST and FashionMNIST datasets.

MNIST	notMNIST	FashionMNIST
	Encoder	
`Linear(784 -> 300)`	`Linear(784 -> 1024)`	`Linear(784 -> 1024)`
`LeakyReLU()`	`LeakyReLU`	`LeakyReLU`
`Linear(300 -> 300)`	`Linear(1024 -> 1024)`	`Linear(1024 -> 1024)`
`LeakyReLU()`	`LeakyReLU`	`LeakyReLU`
$\mu_z \leftarrow$ `Linear(300 -> 40)`	$\mu_z \leftarrow$ `Linear(1024 -> 40)`	$\mu_z \leftarrow$ `Linear(1024 -> 40)`
$\log \sigma_z^2 \leftarrow$ `Linear(300 -> 40)`	$\log \sigma_z^2 \leftarrow$ `Linear(1024 -> 40)`	$\log \sigma_z^2 \leftarrow$ `Linear(1024 -> 40)`
	Decoder	
`Linear(40 -> 300)`	`Linear(40 -> 1024)`	`Linear(40 -> 1024)`
`LeakyReLU()`	`LeakyReLU`	`LeakyReLU`
`Linear(300 -> 300)`	`Linear(1024 -> 1024)`	`Linear(1024 -> 1024)`
`LeakyReLU()`	`LeakyReLU`	`LeakyReLU`
`Linear(300 -> 784)`	`Linear(1024 -> 784)`	`Linear(1024 -> 784)`
$\mu_x \leftarrow$ `Sigmoid()`	$\mu_x \leftarrow$ `Sigmoid()`	$\mu_x \leftarrow$ `Sigmoid()`

Table 6: Convolutional architecture for CelebA dataset.

Encoder	Decoder
`Conv(5x5, 3 -> 32)`	`Linear(128 -> 4096)`
`BatchNorm()`	`ReLU()`
`ReLU()`	`ConvTranspose(5x5, 256 -> 128)`
`Conv(5x5, 32 -> 64)`	`BatchNorm()`
`BatchNorm()`	`ReLU()`
`ReLU()`	`ConvTranspose(5x5, 128 -> 64)`
`Conv(5x5, 64 -> 128)`	`BatchNorm()`
`BatchNorm()`	`ReLU()`
`ReLU()`	`ConvTranspose(5x5, 64 -> 32)`
`Conv(5x5, 128 -> 256)`	`BatchNorm()`
`BatchNorm()`	`ReLU()`
`ReLU()`	`Conv(1x1, 32 -> 3)`
$\mu_z \leftarrow$ `Linear(256 -> 128)`	$\mu_x \leftarrow$ `Softsign()`
$\log \sigma_z^2 \leftarrow$ `Linear(256 -> 128)`	