Recursive Inference for Variational Autoencoders

Minyoung Kim[1]
[1]Samsung AI Center
Cambridge, UK
mikim21@gmail.com

Vladimir Pavlovic[1,2]
[2]Rutgers University
Piscataway, NJ, USA
vladimir@cs.rutgers.edu

Abstract

Inference networks of traditional Variational Autoencoders (VAEs) are typically amortized, resulting in relatively inaccurate posterior approximation compared to instance-wise variational optimization. Recent semi-amortized approaches were proposed to address this drawback; however, their iterative gradient update procedures can be computationally demanding. To address these issues, in this paper we introduce an accurate amortized inference algorithm. We propose a novel recursive mixture estimation algorithm for VAEs that iteratively augments the current mixture with new components so as to maximally reduce the divergence between the variational and the true posteriors. Using the functional gradient approach, we devise an intuitive learning criteria for selecting a new mixture component: the new component has to improve the data likelihood (lower bound) and, at the same time, be as divergent from the current mixture distribution as possible, thus increasing representational diversity. Compared to recently proposed boosted variational inference (BVI), our method relies on amortized inference in contrast to BVI's non-amortized single optimization instance. A crucial benefit of our approach is that the inference at test time requires a single feed-forward pass through the mixture inference network, making it significantly faster than the semi-amortized approaches. We show that our approach yields higher test data likelihood than the state-of-the-art on several benchmark datasets.

1 Introduction

Accurately modeling complex generative processes for high dimensional data (e.g., images) is a key task in deep learning. In many application fields, the Variational Autoencoder (VAE) [13, 29] was shown to be very effective for this task, endowed with the ability to interpret and directly control the latent variables that correspond to underlying hidden factors in data generation, a critical benefit over synthesis-only models such as GANs [7]. The VAE adopts the *inference network* (aka encoder) that can perform test-time inference using a single feed-forward pass through a neural network. Although this feature, known as *amortized inference*, allows VAE to circumvent otherwise time-consuming procedures of solving the instance-wise variational optimization problem at test time, it often results in inaccurate posterior approximation compared to the instance-wise variational optimization [4].

Recently, semi-amortized approaches have been proposed to address this drawback. The main idea is to use an amortized encoder to produce a reasonable initial iterate, followed by instance-wise posterior fine tuning (e.g., a few gradient steps) to improve the posterior approximation [11, 14, 23, 27]. This is similar to the test-time model adaptation of the MAML [5] in multi-task (meta) learning. However, this iterative gradient update may be computationally expensive during both training and test time: for training, some of the methods require Hessian-vector products for backpropagation, while at test time, one has to perform extra gradient steps for fine-tuning the variational optimization. Moreover, the performance of this approach is often very sensitive to the choice of the gradient step size and the number of gradient updates.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

In this paper, we consider a different approach; we build a mixture encoder model, for which we propose a recursive estimation algorithm that iteratively augments the current mixture with a new component encoder so as to reduce the divergence between the resulting variational and the true posteriors. While the outcome is a (conditional) mixture inference model, which could also be estimated by end-to-end gradient descent [34], our recursive estimation method is more effective and less susceptible to issues such as the mixture collapsing. This resiliency is attributed to our specific learning criteria for selecting a new mixture component: the new component has to improve the data likelihood (lower bound) and, at the same time, be as divergent as possible from the current mixture distribution, thus increasing the mixture diversity.

Although a recent family of methods called *Boosted Variational Inference* (BVI) [8, 21, 22, 2, 25] tackles this problem in a seemingly similar manner, our approach differs from BVI in several aspects. Most notably, we address the recursive inference in VAEs in the form of amortized inference, while BVI is developed within the standard VI framework, leading to a non-amortized single optimization instance, inappropriate for VAEs in which the decoder also needs to be simultaneously learned. Furthermore, for the regularization strategy, required in the new component learning stage to avoid degenerate solutions, we employ the *bounded KL loss* instead of the previously used entropy regularization. This approach is better suited for amortized inference network learning in VAEs, more effective as well as numerically more stable than BVI (Sec. 3.1 for detailed discussions).

Another crucial benefit of our approach is that the inference at test time is accomplished using a single feed-forward pass through the mixture inference network, a significantly faster process than the inference in semi-amortized methods. We show that our approach empirically yields higher test data likelihood than standard (amortized) VAE, existing semi-amortized approaches, and even the high-capacity flow-based encoder models on several benchmark datasets.

2 Background

We denote by $\mathbf{x}$ observation (e.g., image) that follows the unknown distribution $p_d(\mathbf{x})$. We aim to learn the VAE model that fits the given iid data $\{\mathbf{x}^i\}_{i=1}^N$ sampled from $p_d(\mathbf{x})$. Specifically, letting $\mathbf{z}$ be the underlying latent vector, the VAE is composed of a prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ and the conditional model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ where the latter, also referred to as the *decoder*, is defined as a tractable density (e.g., Gaussian) whose parameters are the outputs of a deep network with weight parameters $\boldsymbol{\theta}$.

To fit the model, we aim to maximize the data log-likelihood, $\sum_{i=1}^N \log p_{\boldsymbol{\theta}}(\mathbf{x}^i)$ where $p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{p(\mathbf{z})}[p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$. As evaluating the marginal likelihood exactly is infeasible, the variational inference aims to approximate the posterior by a density in some tractable family, that is, $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \approx q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x})$ where $q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x})$ is a tractable density (e.g., Gaussian) with parameters $\boldsymbol{\lambda}$. For instance, if the Gaussian family is adopted, then $q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ constitutes $\boldsymbol{\lambda}$. The approximate posterior $q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x})$ is often called the *encoder*. It is well known that the marginal log-likelihood is lower-bounded by the so-called *evidence lower bound* (ELBO, denoted by $\mathcal{L}$),

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\theta}; \mathbf{x}) := \mathbb{E}_{q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x})}\big[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x})\big], \quad (1)$$

where the gap in (1) is exactly the posterior approximation error $\mathrm{KL}(q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$.

Hence, maximizing $\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\theta}; \mathbf{x})$ with respect to $\boldsymbol{\lambda}$ for the current $\boldsymbol{\theta}$ and the given input instance $\mathbf{x}$, amounts to finding the density in the variational family that best approximates the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. However, notice that the optimum $\boldsymbol{\lambda}$ must be specific to (i.e., dependent on) the input $\mathbf{x}$, and for some other input point $\mathbf{x}'$ one should do the ELBO optimization again to find the optimal encoder parameter $\boldsymbol{\lambda}'$ that approximates the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}')$. The stochastic variational inference (SVI) [9] directly implements this idea, and the approximate posterior inference for a new input point $\mathbf{x}$ in SVI amounts to solving the ELBO optimization on the fly by gradient ascent.

However, the downside is computational overhead since we have to perform iterative gradient ascent to have approximate posterior $q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x})$ for a new input $\mathbf{x}$. To remedy this issue, one can instead consider an ideal function $\boldsymbol{\lambda}^*(\mathbf{x})$ that maps each input $\mathbf{x}$ to the optimal solution $\arg\max_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\theta}; \mathbf{x})$. We then introduce a deep neural network $\boldsymbol{\lambda}(\mathbf{x}; \boldsymbol{\phi})$ with the weight parameters $\boldsymbol{\phi}$ as a universal function approximator of $\boldsymbol{\lambda}^*(\mathbf{x})$. Then the ELBO, now denoted as $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}; \mathbf{x})$, is optimized with respect to $\boldsymbol{\phi}$. This approach, called the *amortized* variational inference (AVI), was proposed in the original VAE [13]. A clear benefit of it is the computational speedup thanks to the feed-forward passing $\boldsymbol{\lambda}(\mathbf{x}; \boldsymbol{\phi})$ used to perform posterior inference for a new input $\mathbf{x}$.

Although AVI is computationally more attractive, it is observed that the quality of data fitting is degraded due to the amortization error, defined as an approximation error originating from the difference between $\boldsymbol{\lambda}^*(\mathbf{x})$ and $\boldsymbol{\lambda}(\mathbf{x};\boldsymbol{\phi})$ [4]. That is, the AVI's computational advantage comes at the expense of reduced approximation accuracy; the SVI posterior approximation can be more accurate since we minimize the posterior approximation error $\text{KL}(q_{\boldsymbol{\lambda}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$ *individually* for each input $\mathbf{x}$. To address this drawback, the *semi-amortized* variational inference (SAVI) approaches have been proposed in [11, 23, 14]. The main idea is to use the amortized encoder to produce a reasonably good initial iterate for the subsequent SVI optimization. The parameters $\boldsymbol{\phi}$ of the amortized encoder are trained in such a way that several steps of warm-start SVI gradient ascent would yield reduction of the instance-wise posterior approximation error, which is similar in nature to the gradient-based meta learning [5] aimed at fast adaptation of the model to a new task in the multi-task meta learning.

However, the iterative gradient update procedure in SAVI is computationally expensive during both training and test times. For training, it requires backpropagation for the objective that involves gradients, implying the need for Hessian evaluation (albeit finite difference approximation). More critically, at test time, the inference requires a time-consuming gradient ascent optimization. Moreover, its performance is often quite sensitive to the choice of the gradient step size and the number of gradient updates; and it is difficult to tune these parameters to achieve optimal performance-efficiency trade-off. Although more recent work [27] mitigated the issue of choosing the step size by the first-order approximate solution method with the Laplace approximation, such linearization of the deep decoder network restricts its applicability to the models containing only fully connected layers, and makes it difficult to be applied to more structured models such as convolutional networks.

3 Recursive Mixture Inference Model (Proposed Method)

Our method is motivated by the premise of the semi-amortized inference (SAVI), i.e., refining the variational posterior to further reduce the difference from the true posterior. However, instead of doing the direct SVI gradient ascent as in SAVI, we introduce another amortized encoder model that augments the first amortized encoder to reduce the posterior approximation error.

Formally, let $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ be our amortized encoder model[1] with the parameters $\boldsymbol{\phi}$. For the current decoder $\boldsymbol{\theta}$, the posterior approximation error $\text{KL}(q(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$ equals -$\mathcal{L}(q,\boldsymbol{\theta};\mathbf{x})$ (up to constant).[2] The goal is to find another amortized encoder model $q'(\mathbf{z}|\mathbf{x})$ with the parameters $\boldsymbol{\phi}'$ such that, when convexly combined with $q(\mathbf{z}|\mathbf{x})$ in a mixture $\epsilon q' + (1-\epsilon)q$ for some small $\epsilon > 0$, the resulting *reduction of the posterior approximation error*, $\Delta\text{KL} := \mathcal{L}(\epsilon q' + (1-\epsilon)q, \boldsymbol{\theta};\mathbf{x}) - \mathcal{L}(q,\boldsymbol{\theta};\mathbf{x})$, is maximized. That is, we seek $\boldsymbol{\phi}'$ that maximizes ΔKL.

Compared to SAVI. The added encoder q' can be seen as the means for correcting q, to reduce the mismatch between q and the true $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. In SAVI, this correction is done by explicit gradient ascent (finetuning) along $\boldsymbol{\phi}$ for every inference query, at train or test time, which is computationally expensive. In contrast, we learn a differential amortized encoder at training time, which is fixed at test time, requiring only a single neural network feed-forward pass to obtain the approximate posterior.

This encoder correction-by-augmentation can continue by regarding the mixture $\epsilon q' + (1-\epsilon)q$ as our current inference model to which another new amortized encoder will be added, with the recursion repeated a few times. This leads to a *mixture* model for the encoder, $Q(\mathbf{z}|\mathbf{x}) = \alpha_0 q(\mathbf{z}|\mathbf{x}) + \alpha_1 q'(\mathbf{z}|\mathbf{x}) + \cdots$, where $\sum_m \alpha_m = 1$. The main question is how to find the next encoder model to augment the current mixture Q. We do this by the functional gradient approach [6, 24].

Functional gradients for mixture component search. Following the functional gradient framework [6, 24], the (ELBO) objective for the mixture $Q(\mathbf{z}|\mathbf{x})$ can be expressed as a *functional*, namely a function that takes a density function Q as input,

$$J(Q) := \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\big[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log Q(\mathbf{z}|\mathbf{x})\big]. \tag{2}$$

Let $Q(\mathbf{z}|\mathbf{x})$ be our current mixture. We aim to find $q(\mathbf{z}|\mathbf{x})$ to be added to Q by convex combination,

$$Q(\mathbf{z}|\mathbf{x}) \leftarrow \epsilon q(\mathbf{z}|\mathbf{x}) + (1-\epsilon)Q(\mathbf{z}|\mathbf{x}) \tag{3}$$

for some small $\epsilon > 0$, that maximizes our objective functional J. To this end we take the functional gradient of the objective $J(Q)$ with respect to Q. For a given input $\mathbf{x}$, we regard the function $Q(\mathbf{z}|\mathbf{x})$

[1] This is a shorthand for $q_{\boldsymbol{\lambda}(\mathbf{x};\boldsymbol{\phi})}(\mathbf{z}|\mathbf{x})$. We often drop the subscript and use $q(\mathbf{z}|\mathbf{x})$ for simplicity in notation.

[2] We often abuse the notation, either $\mathcal{L}(\boldsymbol{\phi},\boldsymbol{\theta};\mathbf{x})$ or $\mathcal{L}(q,\boldsymbol{\theta};\mathbf{x})$ interchangeably.

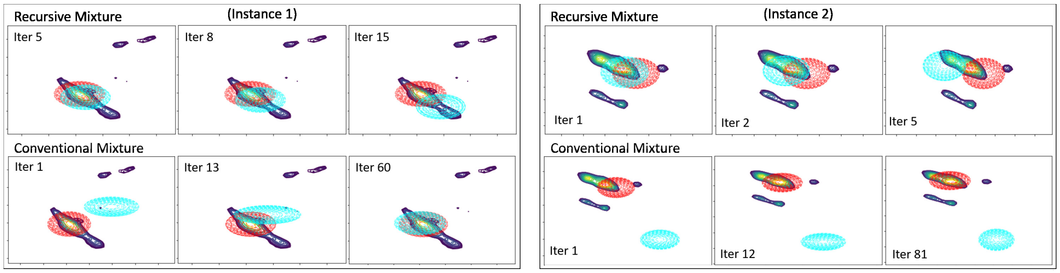

Figure 1: Illustration on MNIST using 2D latent $\mathbf{z}$ space. Results on two data instances (left and right) are shown. (**Top**) Our recursive estimation: The progress of learning the second mixture component is shown from left to right. The contour shows the true posterior $p(\mathbf{z}|\mathbf{x})$, the red is $q_0(\mathbf{z}|\mathbf{x})$, the cyan is the second component that we learn here $q_1(\mathbf{z}|\mathbf{x})$. We only trained q_1; remaining parameters (of the decoder and q_0) are fixed. Parameters of q_1 are initialized to those of q_0. (**Bottom**) Conventional (blind) mixture estimation by end-to-end gradient ascent. For the instance 1 (left), the two components collapse onto each other. For the second (right), a single component (red) becomes dominant while the other (cyan) stays away, unutilized, from the support of the true posterior. The cyan is initialized randomly to be different from the red (otherwise, it constitutes a local minimum).

as an infinite-dimensional vector indexed by $\mathbf{z}$, and take the partial derivative at each $\mathbf{z}$, which yields:

$$\frac{\partial J(Q)}{\partial Q(\mathbf{z}|\mathbf{x})} = \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log Q(\mathbf{z}|\mathbf{x}) - 1. \tag{4}$$

Since we have a convex combination (3), the steepest ascent direction (4) needs to be projected onto the feasible function space $\{q(\cdot|\mathbf{x}) - Q(\cdot|\mathbf{x}) : q \in \mathcal{Q}\}$ where $\mathcal{Q} = \{q_{\boldsymbol{\phi}}\}_{\boldsymbol{\phi}}$ is the set of variational densities realizable by the parameters $\boldsymbol{\phi}$. Formally we solve the following optimization:

$$\max_{q \in \mathcal{Q}} \left\langle q(\cdot|\mathbf{x}) - Q(\cdot|\mathbf{x}), \; \frac{\partial J(Q)}{\partial Q(\cdot|\mathbf{x})} \right\rangle, \tag{5}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in the function space. Using (4), and considering all training samples $\mathbf{x} \sim p_d(\mathbf{x})$, the optimization (5) can be written as:

$$\max_{\boldsymbol{\phi}} \; \mathbb{E}_{p_d(\mathbf{x})} \Big[\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \big[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log Q(\mathbf{z}|\mathbf{x}) \big] \Big], \tag{6}$$

where the outer expectation is with respect to the data distribution $p_d(\mathbf{x})$. By adding and subtracting $\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ to and from the objective, we see that (6) can be rephrased as follows:

$$\max_{\boldsymbol{\phi}} \; \mathbb{E}_{p_d(\mathbf{x})} \Big[\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}; \mathbf{x}) + \mathrm{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) || Q(\mathbf{z}|\mathbf{x})) \Big]. \tag{7}$$

Note that (7) gives us very intuitive criteria of how the new encoder component $q_{\boldsymbol{\phi}}$ should be selected: it has to maximize the ELBO (the first objective term), and at the same time, $q_{\boldsymbol{\phi}}$ should be *different* from the current mixture Q (the KL term). That is, our next encoder has to keep explaining the data well (by large ELBO) while increasing the diversity of the encoder distribution (by large KL), concentrating on those regions of the latent space that were poorly represented by the current Q. This supports our original intuition stated at the beginning of this section. See Fig. 1 for the illustration.

Why recursive estimation. Although we eventually form a (conditional) mixture model for the variational encoder, and such a mixture model can be estimated by end-to-end gradient descent, our recursive estimation is efficient and less susceptible to the known issues of blind mixture estimation, including collapsed mixture components and domination by a single component. This resiliency is attributed to our specific learning criteria for selecting a new mixture component: improve the data likelihood and at the same time be as distinct as possible from the current mixture, thus increasing diversity. See Fig. 1 for an illustrative comparison between our recursive and blind mixture estimation.

3.1 Optimization Strategy

Although we discussed the key idea of recursive mixture estimation, that is, at each step, fixing the current mixture Q and add a new component q, it should be noted that the previously added components q's (and their mixing proportions) need to be refined every time we update the decoder

Algorithm 1 Recursive Learning Algorithm for Mixture Inference Model.

```
Input: Initial $\{q_m(\mathbf{z}|\mathbf{x};\boldsymbol{\phi}_m)\}_{m=0}^M$, $\{\epsilon_m(\mathbf{x};\boldsymbol{\eta}_m)\}_{m=1}^M$, and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. Learning rate $\gamma$. KL bound $C$.
Output: Learned inference and decoder models.
Let: $Q_m = (1-\epsilon_m)Q_{m-1} + \epsilon_m q_m$ $(m = 1 \dots M)$, $Q_0 = q_0$. $\text{BKL}(p||q) = \max(C, \text{KL}(p||q))$.
repeat
  Sample a batch of data $\mathbf{B}$ from $p_d(\mathbf{x})$.
  Update $q_0(\mathbf{z}|\mathbf{x};\boldsymbol{\phi}_0)$:  $\boldsymbol{\phi}_0 \leftarrow \boldsymbol{\phi}_0 + \gamma\nabla_{\boldsymbol{\phi}_0}\mathbb{E}_{\mathbf{x}\sim\mathbf{B}}\big[\mathcal{L}(q_0,\boldsymbol{\theta};\mathbf{x})\big]$.
  for $m = 1,\dots,M$ do
    Update $q_m(\mathbf{z}|\mathbf{x};\boldsymbol{\phi}_m)$:  $\boldsymbol{\phi}_m \leftarrow \boldsymbol{\phi}_m + \gamma\nabla_{\boldsymbol{\phi}_m}\mathbb{E}_{\mathbf{x}\sim\mathbf{B}}\big[\mathcal{L}(q_m,\boldsymbol{\theta};\mathbf{x}) + \text{BKL}(q_m||Q_{m-1})\big]$.
    Update $\epsilon_m(\mathbf{x};\boldsymbol{\eta}_m)$:  $\boldsymbol{\eta}_m \leftarrow \boldsymbol{\eta}_m + \gamma\nabla_{\boldsymbol{\eta}_m}\mathbb{E}_{\mathbf{x}\sim\mathbf{B}}\big[\mathcal{L}\big((1-\epsilon_m)Q_{m-1} + \epsilon_m q_m,\boldsymbol{\theta};\mathbf{x}\big)\big]$.
  end for
  Update $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathbf{x}\sim\mathbf{B}}\big[\mathcal{L}(Q_M,\boldsymbol{\theta};\mathbf{x})\big]$.
until convergence
```

parameters $\boldsymbol{\theta}$. This is due to the VAE framework in which we have to learn the decoder in conjunction with the inference model, one of the main differences from the previous BVI approaches (See Sec. 4).

To this end, we consider a mixture model Q that consists of the *fixed* number (M) of components added to the initial component (denoted by q_0), namely

$$Q(\mathbf{z}|\mathbf{x}) = \alpha_0(\mathbf{x})q_0(\mathbf{z}|\mathbf{x}) + \sum_{m=1}^{M}\alpha_m(\mathbf{x})q_m(\mathbf{z}|\mathbf{x}), \tag{8}$$

where $q_m(\mathbf{z}|\mathbf{x})$ $(m = 0,\dots,M)$ are all amortized encoders whose parameters are denoted by $\boldsymbol{\phi}_m$, and α_m are the mixing proportions. Since the impact of each component can be different from instance to instance, we consider functions $\alpha_m(\mathbf{x})$, instead of scalars. To respect the idea of recursively adding components (i.e., q_m with ϵ_m), the mixing proportions conform to the following implicit structure:

$$\alpha_m(\mathbf{x}) = \epsilon_m(\mathbf{x})\prod_{j=m+1}^{M}(1-\epsilon_j(\mathbf{x})) \;\; \text{for } m = 0,1,\dots,M \;\; (\text{let } \epsilon_0(\mathbf{x}) = 1). \tag{9}$$

This is derived from the recursion, $Q_m = (1-\epsilon_m)Q_{m-1} + \epsilon_m q_m$ for $m = 1,\dots,M$, where we denote by Q_m the mixture formed by $q_0, q_1, \dots, q_m$ with $\epsilon_0(=1), \epsilon_1, \dots, \epsilon_m$, and $Q_0 := q_0$. Hence $Q_M = Q$. Note also that we model $\epsilon_m(\mathbf{x})$ as neural networks $\epsilon_m(\mathbf{x};\boldsymbol{\eta}_m)$ with parameters $\boldsymbol{\eta}_m$.

Now we describe our recursive mixture learning algorithm. As we seek to update all components simultaneously together with the decoder $\boldsymbol{\theta}$, we employ gradient ascent optimization with *all parameters iteratively and repeatedly*. Our algorithm is described in Alg. 1. Notice that for the $\boldsymbol{\phi}$ update in the algorithm, we used the BKL which stands for *Bounded KL*, in place of KL. The KL term in (7) is to be *maximized*, and it can be easily unbounded; In typical situations, $\text{KL}(q||Q)$ can become arbitrarily large by having q concentrate on the region where Q has zero support. To this end, we impose an upper barrier on the KL term, that is, $\text{BKL}(q||Q) = \max(C, \text{KL}(q||Q))$, so that increasing KL beyond the barrier point C gives no incentive. $C = 500.0$ works well empirically.

Similar degeneracy issues have been dealt with in the previous BVI approaches for non-VAE variational inference [8, 21]. Most approaches attempted to regularize small entropy when optimizing the new components to be added. However, the entropy regularization may be less effective for the iterative refinement of the mixture components within the VAE framework, since we have indirect control of the component models (and their entropy values) only through the density parameter networks $\boldsymbol{\lambda}(\mathbf{x};\boldsymbol{\phi})$ in $q_{\boldsymbol{\lambda}(\mathbf{x};\boldsymbol{\phi})}(\mathbf{z}|\mathbf{x})$ (i.e., amortized inference). Furthermore, it encourages the component densities to have large entropy all the time as a side effect, which can lead to a suboptimal solution in certain situations. Our upper barrier method, on the other hand, regularizes the component density only if they are too close (within the range of C KL divergence) to the current mixture, rendering it better chance to find an optimal solution outside the C-ball of the current mixture. In fact, the empirical results in Sec. 5.3 demonstrate that our strategy leads to better performance.

The nested loops in Alg. 1 may appear computationally costly, however, the outer loop usually takes a few epochs (usually no more than 20) since we initialize all components q_m identically with the trained encoder parameters of the standard VAE (afterwards, the components quickly move away from each other due to the BKL term). The mixture order M (the number of the inner iterations) is typically small as well (e.g., between 1 and 4), which renders the algorithm fairly efficient in practice.

4 Related Work

The VAE's issue of amortization error was raised recently [4], and the semi-amortized inference approaches [11, 23, 14] attempted to address the issue by performing the SVI gradient updates at test time. Alternatively one can enlarge the representational capacity of the encoder network, yet still amortized inference. A popular approach is the flow-based models that apply nonlinear invertible transformations to VAE's variational posterior [31, 12]. The transformations could be complex autoregressive mappings, while they can also model full covariance matrices via efficient parametrization to represent arbitrary rotations, i.e., cross-dimensional dependency. Our use of functional gradient in designing a learning objective stems from the framework in [6, 24]. Mathematically elegant and flexible in the learning criteria, the framework was more recently exploited in [3] to unify seemingly different machine learning paradigms. Several mixture-based approaches aimed to extend the representational capacity of the variational inference model. In [33] the variational parameters were mixed with a flexible distribution. In [32] the prior is modeled as a mixture (aggregate posterior), while [17] attempted to tighten the lower bound by matching optimal prior with functional Frank-Wolfe.

Boosted VI. Previously, there were approaches to boost the inference network in variational inference similar to our idea [8, 21, 22, 2, 25], where some of them [21, 22, 2] focused on theoretical convergence analysis, inspired by the Frank-Wolfe [10] interpretation of the greedy nature of the algorithm in the infinite-dimensional (function) space. However, these approaches all aimed for stochastic VI in the non-VAE framework, hence non-amortized inference, whereas we consider amortized inference in the VAE framework in which both the decoder and the inference model need to be learned. We briefly summarize the main differences between the previous BVI approaches and ours as follows: 1) We learn $Q(\mathbf{z}|\mathbf{x})$, a density functional of input $\mathbf{x}$, while BVI optimizes $Q(\mathbf{z})$, a single variational density (not a function of $\mathbf{x}$), and thus involves only single optimization. 2) Within the VAE framework, as the decoder is not optimal in the course of training, we update the decoder and all the inference components iteratively and repeatedly. 3) To avoid degeneracy in KL maximization, we employ the bounded KL instead of BVI's entropy penalization, better suited for amortized inference and more effective in practice. 4) The instant impacts of the components, $\epsilon(\mathbf{x})$ are also modeled input-dependent (as neural networks) rather than tunable scalars as in BVI.

5 Evaluations

We test the proposed recursive inference model[3] on several benchmark datasets. We highlight improved test likelihood scores and reduced inference time, compared to semi-amortized VAEs. We also contrast with flow models that aim to increase modeling accuracy using high capacity encoders.

Competing approaches. VAE: The standard VAE model (amortized inference) [13, 29]. **SA**: The semi-amortized VAE [11]. We fix the SVI gradient step size as 10^{-3}, but vary the number of SVI steps from $\{1, 2, 4, 8\}$. **IAF**: The autoregressive-based flow model for the encoder $q(\mathbf{z}|\mathbf{x})$ [12], which has richer expressiveness than VAE's Gaussian encoder. **HF**: The Householder flow encoder model that represents the full covariance using the Householder transformation [31]. The numbers of flows for IAF and HF are chosen from $\{1, 2, 4, 8\}$. **ME**: For a baseline comparison, we also consider the same mixture encoder model, but unlike our recursive mixture learning, the model is trained conventionally, end-to-end; all mixture components' parameters are updated simultaneously. The number of mixture components is chosen from $\{2, 3, 4, 5\}$. **RME**: Our proposed recursive mixture encoder model. We vary the number of additional components M from $\{1, 2, 3, 4\}$, leading to mixture order 2 to 5. All components are initialized identically with the VAE's encoder. See Supplement for the details.

Datasets. MNIST [19], **OMNIGLOT** [18], **SVHN** [26], and **CelebA** [20]. We follow train/test partitions provided in the data, where 10% of the training sets are randomly held out for validation. For CelebA, we randomly split data into $80\%/10\%/10\%$ train/validation/test sets.

Network architectures. We adopt the convolutional neural networks for the encoder and decoder models for all competing approaches. This is because the convolutional networks are believed to outperform fully connected networks for many tasks in the image domain [16, 30, 28]. We also provide empirical evidence in the Supplement by comparing the test likelihood performance between the two architectures.[4] For the details of the network architectures, refer to the Supplement.

[3] The code is publicly available from `https://github.com/minyoungkim21/recmixvae`

[4] Fully-connected decoder architectures are inferior to the deconvnet when the number of parameters are roughly equal. This is why we exclude comparison with the recent [27], but see Supplement for the results.

Table 1: Test log-likelihood scores estimated by IWAE sampling. The parentheses next to model names indicate: the number of SVI steps in SA, the number of flows in IAF and HF, and the mixture order in ME and RME. The superscripts are the standard deviations. The best (on average) results are boldfaced in **red**. In each column, the statistical significance of the difference between the best model (red) and each competing model, is depicted as color: anything non-colored indicates $p \leq 0.01$ (strongly distinguished), $p \in (0.01, 0.05]$ as yellow-orange, $p \in (0.05, 0.1]$ as orange, $p > 0.1$ as red orange (little evidence of difference) by the Wilcoxon signed rank test. Best viewed in color.

Dataset	MNIST		OMNIGLOT		SVHN		CelebA	
dim($\mathbf{z}$)	20	50	20	50	20	50	20	50
VAE	$930.7^{3.9}$	$1185.7^{3.9}$	$501.6^{1.6}$	$801.6^{4.0}$	$4054.5^{14.3}$	$5363.7^{21.4}$	$12116.4^{25.3}$	$15251.9^{39.7}$
SA$^{(1)}$	$921.2^{2.3}$	$1172.1^{1.8}$	$499.3^{2.5}$	$792.7^{7.9}$	$4031.5^{19.0}$	$5362.1^{35.7}$	$12091.1^{21.6}$	$15285.8^{29.4}$
SA$^{(2)}$	$932.0^{2.4}$	$1176.3^{3.4}$	$501.0^{2.7}$	$793.1^{4.8}$	$4041.5^{15.5}$	$5377.0^{23.2}$	$12087.1^{21.5}$	$15252.7^{29.0}$
SA$^{(4)}$	$925.5^{2.6}$	$1171.3^{3.5}$	$488.2^{1.8}$	$794.4^{1.9}$	$4051.9^{22.2}$	$5391.7^{20.4}$	$12116.3^{20.5}$	$15187.3^{27.9}$
SA$^{(8)}$	$928.1^{3.9}$	$1183.2^{3.4}$	$490.3^{2.8}$	$799.4^{2.7}$	$4041.6^{9.5}$	$5370.8^{18.5}$	$12100.6^{22.8}$	$15096.5^{27.2}$
IAF$^{(1)}$	$934.0^{3.3}$	$1180.6^{2.7}$	$489.9^{1.9}$	$788.8^{4.1}$	$4050.0^{9.4}$	$5368.3^{11.5}$	$12098.0^{20.6}$	$15271.2^{28.6}$
IAF$^{(2)}$	$931.4^{3.7}$	$1190.1^{1.9}$	$494.9^{1.4}$	$795.7^{2.7}$	$4054.6^{10.5}$	$5360.0^{10.0}$	$12104.5^{21.8}$	$15262.2^{27.8}$
IAF$^{(4)}$	$926.3^{2.6}$	$1178.1^{1.6}$	$496.0^{2.0}$	$775.1^{2.2}$	$4048.6^{8.7}$	$5338.1^{10.2}$	$12094.6^{22.6}$	$15261.0^{28.1}$
IAF$^{(8)}$	$934.1^{2.4}$	$1150.0^{2.2}$	$498.8^{2.3}$	$774.7^{2.9}$	$4042.0^{9.6}$	$5341.8^{10.1}$	$12109.3^{22.0}$	$15241.5^{27.9}$
HF$^{(1)}$	$917.2^{2.6}$	$1204.3^{4.0}$	$488.6^{2.0}$	$795.9^{3.3}$	$4028.8^{9.7}$	$5372.0^{10.1}$	$12077.2^{31.4}$	$15240.5^{27.6}$
HF$^{(2)}$	$923.9^{3.1}$	$1191.5^{10.8}$	$495.9^{1.8}$	$784.5^{4.8}$	$4030.7^{9.9}$	$5376.6^{10.2}$	$12093.0^{25.6}$	$15258.2^{30.3}$
HF$^{(4)}$	$927.3^{2.8}$	$1197.2^{1.5}$	$487.0^{2.7}$	$799.7^{3.2}$	$4038.4^{9.7}$	$5371.8^{9.8}$	$12082.0^{27.0}$	$15266.5^{29.5}$
HF$^{(8)}$	$928.5^{3.1}$	$1184.1^{1.8}$	$488.3^{2.4}$	$794.6^{4.0}$	$4035.9^{8.9}$	$5351.1^{11.1}$	$12087.3^{25.5}$	$15248.7^{29.7}$
ME$^{(2)}$	$926.7^{3.0}$	$1152.8^{1.7}$	$491.7^{1.4}$	$793.4^{3.8}$	$4037.2^{11.0}$	$5343.2^{13.1}$	$12072.7^{23.3}$	$15290.5^{29.3}$
ME$^{(3)}$	$933.1^{4.1}$	$1162.8^{4.7}$	$491.2^{2.1}$	$807.5^{4.9}$	$4053.8^{16.1}$	$5367.7^{15.8}$	$12100.3^{21.7}$	$15294.6^{28.3}$
ME$^{(4)}$	$914.7^{2.3}$	**$1205.1^{2.3}$**	$491.3^{1.8}$	$732.0^{3.1}$	$4061.3^{12.0}$	$5191.9^{18.5}$	$12092.2^{22.6}$	$15270.7^{20.6}$
ME$^{(5)}$	$920.6^{1.9}$	$1198.5^{3.5}$	$478.0^{2.8}$	$805.7^{3.8}$	$4057.5^{12.2}$	$5209.2^{12.8}$	$12095.3^{25.1}$	$15268.8^{27.5}$
RME$^{(2)}$	$943.9^{1.6}$	$1201.7^{0.9}$	$508.2^{1.2}$	**$821.0^{3.1}$**	$4085.3^{9.7}$	$5403.2^{10.2}$	$12193.1^{23.5}$	$15363.0^{31.7}$
RME$^{(3)}$	$945.1^{1.6}$	$1202.4^{1.0}$	$507.5^{1.1}$	$820.4^{0.9}$	$4085.9^{9.8}$	$5405.1^{10.4}$	$12192.3^{23.5}$	$15365.6^{31.4}$
RME$^{(4)}$	**$945.2^{1.6}$**	$1203.1^{1.0}$	$509.0^{1.2}$	$819.9^{0.9}$	$4080.7^{9.9}$	$5403.8^{10.2}$	$12192.6^{23.4}$	$15364.3^{31.5}$
RME$^{(5)}$	$945.0^{1.7}$	$1203.7^{1.0}$	**$509.1^{1.4}$**	$819.9^{0.9}$	**$4086.9^{10.9}$**	**$5405.5^{8.5}$**	**$12194.2^{11.5}$**	**$15366.2^{12.7}$**

Experimental setup. We vary the latent dim($\mathbf{z}$), small (20) or large (50).[5] To report the test log-likelihood scores $\log p(\mathbf{x})$, we use the importance weighted sampling estimation (IWAE) method [1] with 100 samples (Supplement for details). For each model/dataset, we perform 10 runs with different random train/validation splits, where each run consists of three trainings by starting with different random model parameters, among which only one model with the best validation result is chosen.

Table 2: Test data log-likelihood scores for the **Binary MNIST**. Our results are in the column titled "CNN". The column "FC" is excerpted from [27].

	CNN	FC
VAE	-84.49	-85.38
SA$^{(1)}$	-83.64	-85.20
SA$^{(2)}$	-83.79	-85.10
SA$^{(4)}$	-83.85	-85.43
SA$^{(8)}$	-84.02	-85.24
IAF$^{(1)}$	-83.37	-84.26
IAF$^{(2)}$	-83.15	-84.16
IAF$^{(4)}$	-83.08	-84.03
IAF$^{(8)}$	-83.12	-83.80
HF$^{(1)}$	-83.82	-85.27
HF$^{(2)}$	-83.70	-85.31
HF$^{(4)}$	-83.87	-85.22
HF$^{(8)}$	-83.76	-85.41
ME$^{(2)}$	-83.77	-
ME$^{(3)}$	-83.81	-
ME$^{(4)}$	-83.83	-
ME$^{(5)}$	-83.75	-
VLAE$^{(2)}$	-	-83.72
VLAE$^{(3)}$	-	-83.84
VLAE$^{(4)}$	-	-83.73
VLAE$^{(5)}$	-	-83.60
RME$^{(2)}$	-83.14	-
RME$^{(3)}$	-83.14	-
RME$^{(4)}$	-83.09	-
RME$^{(5)}$	-83.15	-

5.1 Results

The test log-likelihood scores are summarized in Table 1.[6] Overall the results indicate that our recursive mixture encoder (RME) outperforms the competing approaches consistently for all datasets. To see the statistical significance, we performed the one-sided Wilcoxon signed rank test for every pair (the best model, non-best model). The results indicate that this superiority is statistically significant.

Comparison to ME. With one exception, specifically ME (4) with dim($\mathbf{z}$) $=$ 50 on the MNIST, the blind end-to-end mixture learning (ME) consistently underperforms our RME. As also illustrated in Fig. 1, the blind mixture estimation can potentially suffer from mixture collapsing and single dominant component issues. The fact that even the VAE often performs comparably to the ME with different mixture orders supports this observation. On the other hand, our recursive mixture estimation is more robust to the initial parameters. Due to its incremental learning nature, it "knows" the regions in the latent space ill-represented by the current mixture, then updates mixture components to complement those regions. This strategy allows the RME to effectively model highly multi-modal posterior distributions, yielding more robust and accurate variational posterior approximation.

Comparison to SA. The semi-amortized approach (SA) sometimes achieves improvement over the VAE, but not consistently. In particular, its performance

[5] The results for dim($\mathbf{z}$) $= 10$ and 100, also on the **CIFAR10** dataset [15], are reported in the Supplement.

[6] The MNIST results mismatch those reported in the related work (e.g., [32]). Significantly higher scores. This is because we adopt the Gaussian decoder models, not the binary decoders, for all competing methods.

is generally very sensitive to the number of SVI gradient update steps. This is another drawback of the SA, where the gradient-based adaption has to be performed at the test time. Although one could adjust the gradient step size (in place of currently used fixed step size) to improve the performance, there is little principled way to tune the step size at test time that can attain optimal accuracy and inference time trade off. The number of SVI steps in the SA may correspond to the mixture order in our RME model, and the results show that increasing the mixture order usually improves, and not deteriorate, the generalization performance.

Comparison to IAF/HF. Although flow models have rich representational capacity, possibly with full covariance matrices (HF), the improvement over the VAE is limited compared to our RME; the models sometimes perform not any better than the VAE. The failure of the flow-based models may originate from the difficulty of optimizing the complex encoder models. (Similar observations were made in related previous work [27]). This result signifies that sophisticated and discriminative learning criteria are critical, beyond just enlarging the structural capacity of the neural networks, similarly observed from the failure of conventional mixtures.

Non-Gaussian likelihood model. Our empirical evaluations were predominantly conducted with the convolutional architectures on real-valued image data. For the performance of our model with non-convolutional (fully connected) network architectures, the readers can refer to Table 5 and 6 in the supplementary material. For the binarized input images, we have conducted extra experiments on the **Binary MNIST** dataset. The binary images can be modeled by a Bernoulli likelihood in the decoder. Table 2 summarized the results. We have set the latent dimension $\dim(\mathbf{z}) = 50$, and used the same CNN architectures as before, except that the decoder output is changed from Gaussian to Bernoulli. We also include the reported results from [27] for comparison, which employed the same latent dimension 50 and fully connected encoder/decoder networks with similar model complexity as our CNNs'. As shown, IAF and our RME performs equally the best, although the performance differences among the competing approaches are not very pronounced compared to real-valued image cases.

5.2 Test Inference Time

Another key advantage of our recursive mixture inference is the computational efficiency of test-time inference, comparable to that of VAE. Unlike the semi-amortized approaches, where one performs the SVI gradient adaptation at test time, the inference in our RME is merely a single feed forward pass through our mixture encoder network. That is, once training is done, our mixture inference model remains fixed, with no adaptation required.

To verify this empirically, we measure the actual inference time for the competing approaches. The per-batch test inference times (batch size 128) on all benchmark datasets are shown in Tab. 8. To report the results, for each method and each dataset, we run the inference over the entire test set batches, measure the running time, then take the per-batch average. We repeat the procedure five times and report the average. All models are run on the same machine with a single GPU (RTX 2080 Ti), Core i7 3.50GHz CPU, and 128 GB RAM. While we only report test times for $\dim(\mathbf{z}) = 50$, the impact of the latent dimension appears to be less significant.

Table 3: Inference time (milliseconds).

	MNIST	OMNIG.	SVHN	CELEBA
VAE	3.6	4.8	2.2	2.7
SA$^{(1)}$	9.7	11.6	7.0	8.4
SA$^{(2)}$	18.1	19.2	15.5	13.8
SA$^{(4)}$	32.2	34.4	30.1	27.1
SA$^{(8)}$	60.8	65.7	60.3	53.8
IAF$^{(1)}$	4.8	5.7	3.4	4.4
IAF$^{(2)}$	5.9	6.4	3.7	5.1
IAF$^{(4)}$	6.2	7.0	4.7	5.7
IAF$^{(8)}$	7.7	8.2	5.7	7.7
RME$^{(2)}$	4.7	5.4	3.2	4.2
RME$^{(3)}$	4.9	5.5	3.6	4.1
RME$^{(4)}$	4.6	5.3	3.5	4.2
RME$^{(5)}$	4.8	5.6	3.3	4.8

As expected, the semi-amortized approach suffers from the computational overhead of test-time gradient updates, with the inference time significantly increased as the number of updates increases. Our RME is comparable to VAE, and faster than IAF (with more than a single flow), which verifies our claim. Interestingly, increasing the mixture order in our model rarely affects the inference time, due to intrinsic parallelization of the feed forward pass through the multiple mixture components networks, leading to inference time as fast as that of VAE.

5.3 Comparison with Boosted VI's Entropy Regularization

Recall that our RME adopted the bounded KL (BKL) loss to avoid degeneracy in the component update stages. Previous boosted VI (BVI) approaches employ different regularization, namely penalizing small entropy for the new components. However, such indirect regularization can be

Table 4: Comparison with the BVI's entropy regularization [21]. The same color scheme as Tab. 1.

Dataset	MNIST		OMNIGLOT		SVHN		CelebA	
dim($\mathbf{z}$)	20	50	20	50	20	50	20	50
RME$^{(2)}$	$943.9^{1.6}$	$1201.7^{0.9}$	$508.2^{1.2}$	$\mathbf{821.0}^{3.1}$	$4085.3^{9.7}$	$5403.2^{10.2}$	$12193.1^{23.5}$	$15363.0^{31.7}$
RME$^{(3)}$	$945.1^{1.6}$	$1202.4^{1.0}$	$507.5^{1.1}$	$820.4^{0.9}$	$4085.9^{9.8}$	$5405.1^{10.4}$	$12192.3^{23.5}$	$15365.6^{31.4}$
RME$^{(4)}$	$\mathbf{945.2}^{1.6}$	$1203.1^{1.0}$	$509.0^{1.2}$	$819.9^{0.9}$	$4080.7^{9.9}$	$5403.8^{10.2}$	$12192.6^{23.4}$	$15364.3^{31.5}$
RME$^{(5)}$	$945.0^{1.7}$	$\mathbf{1203.7}^{1.0}$	$\mathbf{509.1}^{1.4}$	$819.9^{0.9}$	$\mathbf{4086.9}^{10.9}$	$\mathbf{5405.5}^{8.5}$	$\mathbf{12194.2}^{11.5}$	$\mathbf{15366.2}^{12.7}$
BVI$^{(2)}$	$939.7^{2.8}$	$1196.2^{2.8}$	$507.9^{2.2}$	$817.1^{3.3}$	$4077.3^{10.3}$	$5388.2^{10.2}$	$12133.5^{25.1}$	$15206.4^{28.2}$
BVI$^{(3)}$	$939.5^{2.9}$	$1191.6^{2.9}$	$507.8^{2.2}$	$816.6^{3.4}$	$4076.6^{10.3}$	$5384.2^{10.5}$	$12146.5^{22.4}$	$15249.5^{28.1}$
BVI$^{(4)}$	$937.8^{2.9}$	$1191.6^{2.8}$	$507.8^{2.3}$	$816.8^{3.4}$	$4073.1^{10.2}$	$5371.1^{10.4}$	$12127.7^{22.3}$	$15085.8^{28.4}$
BVI$^{(5)}$	$931.2^{3.0}$	$1183.1^{2.9}$	$508.2^{2.3}$	$816.4^{3.3}$	$4071.2^{10.2}$	$5378.1^{10.1}$	$12092.3^{22.3}$	$15052.5^{28.0}$

less effective for the iterative refinement of the mixture components within the VAE framework (the second last paragraph of Sec. 3.1). To verify this claim, we test our RME models with the BKL loss replaced by the BVI's entropy regularization. More specifically, following the scheme of [21], we replace our BKL loss by $\nu \cdot \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[-\log q(\mathbf{z}|\mathbf{x})]$ estimated by Monte Carlo, where $\nu = 1/\sqrt{t+1}$ is the impact that decreases as the training iteration t.[7] See Tab. 4 for the results. This empirical result demonstrates that our bounded KL loss consistently yields better performance than entropy regularization. We also observe that our BKL loss leads to numerically more stable solutions: For entropy regularization, we had to reduce the learning rate to the tenth of that of BKL to avoid NaNs.

6 Conclusion

In this work we addressed the challenge of improving traditional, amortized inference in VAEs using a mixture of inference networks approach. We demonstrated that this method is both effective in increasing the accuracy of inference and computationally efficient, compared to state-of-the-art semi-amortized inference approaches. This is, in part, due to the effectiveness of the functional recursive mixture learning algorithm we devise and the nature of the inference model, which does not need to be adapted during the test phase. As a consequence, our approach yields higher test data likelihood than the competing approaches on several benchmark datasets, but remains as computationally efficient as the conventional VAE inference. Our recursive model currently requires users to supply the mixture order as an input to the algorithm. In our future work, we aim to investigate principled ways of selecting the mixture order (i.e., model augmentation stopping criteria). We also seek to apply our model to domains with structured data, including sequences (e.g., videos, natural language sentences) and graphs (e.g., molecules, 3D shapes).

Broader Impact

1. **Who may benefit from this research?** For any individuals, practitioners, organizations, and groups who aim to identify the underlying generative process of the high-dimensional structured data via the variational auto-encoding model framework, this research can be a very useful tool that provides highly accurate solutions generalizable to unseen data.
2. **Who may be put at disadvantage from this research?** Not particularly applicable.
3. **What are the consequences of failure of the system?** Any failure of the system that implements our algorithm would not do any serious harm since the failure can be easily detectable at the validation stage, in which case alternative strategies or internal decisions might be looked for.
4. **Whether the task/method leverages biases in the data?** Our method does not leverage biases in the data.

[7] We also tested a slight variant, [8]'s closed-form Gaussian entropy $\log \det \mathbf{\Sigma}$ where $\mathbf{\Sigma}$ is the (diagonal) covariance of the new component $q(\mathbf{z}|\mathbf{x})$. The results were very similar to the scheme of [21]. See Supplement.

Supplementary Material

This supplement consists of the following materials:

7 Detailed Experimental Setups

7.1 Competing Approaches

The competing approaches are summarized as follows:

- **VAE**: The standard VAE model (amortized inference) [13, 29].
- **SA**: The semi-amortized VAE [11]. We fix the SVI gradient step size as 10^{-3}, but vary the number of SVI steps from $\{1, 2, 4, 8\}$.
- **IAF**: The autoregressive-based flow model for the encoder $q(\mathbf{z}|\mathbf{x})$ [12], which has richer expressiveness than VAE's post-Gaussian encoder. The number of flows is chosen from $\{1, 2, 4, 8\}$.
- **HF**: The Householder flow encoder model that represents the full covariance using the Householder transformation [31]. The number of flows is chosen from $\{1, 2, 4, 8\}$.
- **ME**: For a baseline comparison, we also consider the same mixture encoder model, but unlike our recursive mixture learning, the model is trained conventionally, end-to-end; all mixture components' parameters are updated simultaneously. The number of mixture components is chosen from $\{2, 3, 4, 5\}$.
- **RME**: Our proposed recursive mixture encoder model. We vary the number of the components to be added M from $\{1, 2, 3, 4\}$, leading to mixture order 2 to 5.

In addition, we test our RME model modified to employ the previous Boosted VI's entropy regularization schemes. More specifically, we replace our bounded KL loss with the two entropy regularization methods as follows:

- **BVI-ER1**: Following [21], we replace our bounded KL loss by $\nu \cdot \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[-\log q(\mathbf{z}|\mathbf{x})]$ estimated by Monte Carlo, where $\nu = 1/\sqrt{t+1}$ is the impact that decreases as the training iteration t.
- **BVI-ER2**: Instead of the Monte Carlo estimation of the entropy, we use [8]'s closed-form Gaussian entropy $\log\det\mathbf{\Sigma}$ where $\mathbf{\Sigma}$ is the (diagonal) covariance of the new component $q(\mathbf{z}|\mathbf{x})$.

7.2 Datasets

The following benchmark datasets are used. We randomly hold out 10% of the training data as validation sets, except for **CelebA**.

- **MNIST** [19]: $60,000$ training images and $10,000$ test images where each image is of dimension $(28 \times 28 \times 1)$.
- **OMNIGLOT** [18]: $24,345$ training images and $8,070$ test images where each image is of dimension $(28 \times 28 \times 1)$.

- **CIFAR10** [15]: $50,000$ training images and $10,000$ test images where each image is of dimension $(32 \times 32 \times 3)$.
- **SVHN** [26]: $73,257$ training images and $26,032$ test images where each image is of dimension $(32 \times 32 \times 3)$.
- **CelebA** [20]: $202,599$ tightly cropped face images of size $(64 \times 64 \times 3)$. We randomly split the data into $80\%/10\%/10\%$ train/validation/test sets.

7.3 Network Architectures

We adopt the convolutional neural networks for both the encoder and decoder models for all competing approaches. This is because the convolutional networks are believed to outperform fully connected networks for many tasks in the image domain [16, 30, 28]. We also provide empirical evidence in Sec. 9 of this Supplement that the fully-connected decoder architecture is inferior to the deconvnet decoder that we adopted, when the two architectures have roughly equal numbers of parameters. This is why we excluded comparison with the recent Laplacian approximation approach of [27] in the main paper. They use the first-order approximate solver method to obtain the mode of the true posterior, but such linearization of a deep network is only computationally feasible for *fully connected* decoder models. On the other hand, our recursive mixture learning admits arbitrary types of encoder/decoder architectures, which is another advantage. In Sec. 9 of this Supplement we empirically compare the performance between the Laplace approximation [27] and our approach.

For the encoder architecture, we first apply L convolutional layers with (4×4)-pixels kernels, followed by two fully-connected layers with hidden layers dimension h. For the decoder, the input images first go through two fully connected layers, followed by L deconvolution (*transposed convolution*) layers with (4×4)-pixels filters. Here, $L = 3$ for all datasets except CelebA which has $L = 4$. The hidden layer dimension $h = 256$ for MNIST/OMNIGLOT and $h = 512$ for the others. For fair comparison, the same convolutional network architectures are used in all competing methods.

For our recursive mixture RME, all mixture components of the inference model are initialized identically with the VAE's encoder. For the ME (blind end-to-end mixture learning), the first mixture component is initialized with the VAE's encoder while the others are chosen randomly. This is because initializing all components identically would constitute a local maximum of the log-likelihood objective function of the ME, making it unable to update the model further. For the IAF, we follow the inverse autoregressive flow modeling [12] where we use the two-layer MADE [**?**] (with the number of hidden units 500) as the autoregressiveNN network. The base density, which is transformed to a more complex density by the flow, is initialized with the trained VAE's encoder $q(\mathbf{z}|\mathbf{x})$. For the HF, the latents of the base encoder go through a number of linear transformations, followed by the Householder transformation, where the base encoder is also initialized with the VAE's encoder.

The decoder is modeled as transposed convolutional networks. The network architectures are slightly different across the datasets due to different input image dimensions. We summarize the full network architectures in Tab. 5 (MNIST and OMNIGLOT), Tab. 6 (CIFAR10 and SVHN), and Tab. 7 (CelebA).

In our recursive mixture model, we also need to define the impact function $\epsilon(\mathbf{x})$ for each component. We used a fully connected network $\epsilon(\mathbf{x}; \boldsymbol{\eta})$ with one hidden layer of dimension 10. To prevent a new component from overly taking the mixing proportion, we set an upper bound $\epsilon_{\max}$ on the output of the network. This is done by applying the sigmoid function to the output of $\epsilon(\mathbf{x})$, and multiplication by $\epsilon_{\max}$. For all our experiments $\epsilon_{\max} = 0.1$ worked well.

7.4 Experimental Setups

For all optimization, we used the Adam optimizer with batch size 128 and learning rate 0.0005. We run the optimization until 2000 epochs. We vary the latent dimension $\dim(\mathbf{z})$, from $\{10, 20, 50, 100\}$. To report the test log-likelihood scores $\log p(\mathbf{x})$, we use the importance weighted sampling estimation (IWAE) method [1]. More specifically,

$$\text{IWAE} = \log \left(\frac{1}{K} \sum_{i=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_i)}{q(\mathbf{z}_i|\mathbf{x})} \right), \tag{10}$$

Table 5: Encoder (i.e., each component in our mixture model) and decoder network architectures for MNIST and OMNIGLOT datasets. In the convolutional and transposed convolutional layers, the paddings are properly adjusted to match the input/output dimensions.

Encoder	Decoder
Input: $(28 \times 28 \times 1)$	Input: $\mathbf{z} \in \mathbb{R}^p$ $(p \in \{10, 20, 50, 100\})$
32 (4×4) conv.; stride 2; LeakyReLU (0.01)	FC. 256; ReLU
32 (4×4) conv.; stride 2; LeakyReLU (0.01)	FC. $3 \cdot 3 \cdot 64$; RELU
64 (4×4) conv.; stride 2; LeakyReLU (0.01)	32 (4×4) Transposed Conv.; stride 2; ReLU
FC. 256; LeakyReLU (0.01)	32 (4×4) Transposed Conv.; stride 2; ReLU
FC. 2 $\times p$ $(p = \text{dim}(\mathbf{z}) \in \{10, 20, 50, 100\})$	1 (4×4) Transposed Conv.; stride 2

Table 6: Encoder and decoder network architectures for CIFAR10 and SVHN datasets.

Encoder	Decoder
Input: $(32 \times 32 \times 3)$	Input: $\mathbf{z} \in \mathbb{R}^p$ $(p \in \{10, 20, 50, 100\})$
32 (4×4) conv.; stride 2; LeakyReLU (0.01)	FC. 512; ReLU
32 (4×4) conv.; stride 2; LeakyReLU (0.01)	FC. $4 \cdot 4 \cdot 64$; RELU
64 (4×4) conv.; stride 2; LeakyReLU (0.01)	32 (4×4) Transposed Conv.; stride 2; ReLU
FC. 512; LeakyReLU (0.01)	32 (4×4) Transposed Conv.; stride 2; ReLU
FC. 2 $\times p$ $(p = \text{dim}(\mathbf{z}) \in \{10, 20, 50, 100\})$	3 (4×4) Transposed Conv.; stride 2

where $\mathbf{z}_1, \ldots, \mathbf{z}_K$ are i.i.d. samples from $q(\mathbf{z}|\mathbf{x})$. It can be shown that IWAE lower bounds $\log p(\mathbf{x})$ and can be arbitrarily close to the target as the number of samples K grows. We use $K = 100$ throughout the experiments.

For each model/dataset, we perform 10 runs with different random train/validation splits, where each run consists of three trainings by starting with different random model parameters, among which only one model with the highest validation performance is chosen. To see the statistical significance of difference between competing models, we also performed the one-sided Wilcoxon signed rank test for every pair, namely (the best model vs. each non-best model), using the 10 log-likelihood scores per model.

Table 7: Encoder and decoder network architectures for CelebA dataset.

Encoder	Decoder
Input: $(64 \times 64 \times 3)$	Input: $\mathbf{z} \in \mathbb{R}^p$ $(p \in \{10, 20, 50, 100\})$
32 (4×4) conv.; stride 2; LeakyReLU (0.01)	FC. 512; ReLU
32 (4×4) conv.; stride 2; LeakyReLU (0.01)	FC. $4 \cdot 4 \cdot 64$; RELU
64 (4×4) conv.; stride 2; LeakyReLU (0.01)	64 (4×4) Transposed Conv.; stride 2; ReLU
64 (4×4) conv.; stride 2; LeakyReLU (0.01)	32 (4×4) Transposed Conv.; stride 2; ReLU
FC. 512; LeakyReLU (0.01)	32 (4×4) Transposed Conv.; stride 2; ReLU
FC. 2 $\times p$ $(p = \text{dim}(\mathbf{z}) \in \{10, 20, 50, 100\})$	3 (4×4) Transposed Conv.; stride 2

Table 8: (Per-batch) Test inference time (in milliseconds) with batch size 128. The latent dimension $\dim(\mathbf{z}) = 50$.

	MNIST	OMNIG.	CIFAR10	SVHN	CelebA
VAE	3.6	4.8	3.7	2.2	2.7
SA (1)	9.7	11.6	9.8	7.0	8.4
SA (2)	18.1	19.2	16.8	15.5	13.8
SA (4)	32.2	34.4	27.9	30.1	27.1
SA (8)	60.8	65.7	60.5	60.3	53.8
IAF (1)	4.8	5.7	5.1	3.4	4.4
IAF (2)	5.9	6.4	5.6	3.7	5.1
IAF (4)	6.2	7.0	6.3	4.7	5.7
IAF (8)	7.7	8.2	7.6	5.7	7.7
RME (2)	4.7	5.4	4.9	3.2	4.2
RME (3)	4.9	5.5	5.1	3.6	4.1
RME (4)	4.6	5.3	5.1	3.5	4.2
RME (5)	4.8	5.6	5.1	3.3	4.8

8 Experimental Results

The test log-likelihood scores are summarized in Tab. 11 (MNIST)[8], Tab. 12 (OMNIGLOT), Tab. 13 (CIFAR10), Tab. 14 (SVHN), and Tab. 15 (CelebA). We also report the performance of the entropy regularization schemes introduced in the previous Boosted VI (BVI) approaches. To this end, in our RME, we replace our bounded KL (BKL) loss with the entropy regularization. More specifically, we consider two entropy regularization schemes – **BVI-ER1**: [21]'s regularization of the negative entropy of $q(\mathbf{z}|\mathbf{x})$ whose impact decreases $\frac{1}{\sqrt{t+1}}$ as a function of training iteration t, as suggested. **BVI-ER2**: [8]'s Gaussian entropy based regularization (i.e., penalizing small $\log\det\mathbf{\Sigma}$ where $\mathbf{\Sigma}$ is the (diagonal) covariance matrix of the new component $q(\mathbf{z}|\mathbf{x})$ to be optimized. Overall the results indicate that our recursive mixture encoder (RME) outperforms the competing approaches consistently for all datasets.

8.1 Test Inference Time

Another key advantage of our recursive mixture model is the computational efficiency of test-time inference, comparable to that of VAE. Unlike the semi-amortized approaches, where one performs the SVI gradient adaptation at test time, the inference in our RME is merely a single feed forward pass through our mixture encoder network. That is, once training is done, our mixture inference model remains fixed, with no adaptation required.

To verify this, we measure the actual inference time for competing approaches. The per-batch inference times (batch size 128) on all benchmark datasets are shown in Tab. 8. To report the results, for each method and each dataset, we run the inference over the entire test set batches, measure the running time, then take the per-batch average. We repeat the procedure five times and report the average. All models are run on the same machine with a single GPU (RTX 2080 Ti), Core i7 3.50GHz CPU, and 128 GB RAM. We only report test times for the latent dimension $\dim(\mathbf{z}) = 50$ as the impact of the latent dimension appears to be less significant.

As expected, the semi-amortized approach (SA) suffers from the computational overhead of test time gradient updates, with the inference time significantly increased as the number of the updates increases. Our RME is comparable to the VAE, and faster than the IAF (with more than a single flow), which verifies our claim. Interestingly, increasing the mixture order in our model rarely affects the inference time, due to intrinsic parallelization of the feed forward pass through the multiple mixture components networks, leading to inference times as fast as those of the single component model (VAE).

[8] For the MNIST results, the test log-likelihood scores of the competing methods mismatch those reported in the related work (e.g., [32]). Significantly higher scores. This is because we adopt the Gaussian decoder models, not the binary decoders, for all competing methods.

Table 9: (Fully connected vs. convolutional decoder networks) Test log-likelihood scores (unit in nat). The figures without parentheses are the scores using the fully connected networks, whereas figures in the parentheses are the scores using the convolutional decoder networks. Both architectures have roughly equal number of the weight parameters. The number of linearization steps in the VLAE is chosen from $\{1, 2, 4, 8\}$.

	MNIST		OMNIGLOT	
	$\textsc{dim}(\mathbf{z}) = 10$	$\textsc{dim}(\mathbf{z}) = 50$	$\textsc{dim}(\mathbf{z}) = 10$	$\textsc{dim}(\mathbf{z}) = 50$
VAE	563.6 (685.1)	872.6 (1185.7)	296.8 (347.0)	519.4 (801.6)
SA (1)	565.1 (688.1)	865.8 (1172.1)	297.6 (344.1)	489.0 (792.7)
SA (2)	565.3 (682.2)	868.2 (1176.3)	295.3 (349.5)	534.1 (793.1)
SA (4)	565.9 (683.5)	852.9 (1171.3)	294.8 (342.1)	497.8 (794.4)
SA (8)	564.9 (684.6)	870.9 (1183.2)	299.0 (344.8)	500.0 (799.4)
VLAE (1)	590.0	922.2	307.4	644.0
VLAE (2)	595.1	908.8	307.6	621.4
VLAE (4)	605.2	841.4	318.0	597.7
VLAE (8)	605.7	779.9	316.6	553.1
RME (2)	570.9 (697.2)	888.1 (1201.7)	298.4 (349.3)	524.7 (821.0)
RME (3)	571.9 (698.2)	888.2 (1202.4)	298.6 (349.9)	524.8 (820.4)
RME (4)	571.4 (699.0)	888.1 (1203.1)	298.8 (350.7)	525.3 (819.9)
RME (5)	572.2 (699.4)	888.0 (1203.7)	298.8 (351.1)	526.9 (819.9)

9 Comparison with Fully-Connected Decoder Networks

In the main paper we used the convolutional networks for both encoder and decoder models. This is a reasonable architectural choice considering that all the datasets are images. Also it is widely believed that convolutional networks outperform fully connected networks for many tasks in the image domain [16, 30, 28]. However, one can alternatively consider fully connected networks for either the encoder or the decoder, or both. Nevertheless, being equal in the number of model parameters, using both convolutional encoder and decoder networks always outperformed the fully connected counterparts. In this section we empirically verify this by comparing the test likelihood performance between the two architectures. We particularly focus on comparing the two architectures (convolutional vs. fully connected) for the *decoder* model alone, while retaining the convolutional network encoder for both cases.

Using the fully connected decoder network allows us to test the recent Laplacian approximation approach [27] (denoted by **VLAE**), which we excluded from the main paper. They employ a first-order approximation solver to find the mode of the true posterior (i.e., linearizing the decoder function), and compute the Hessian of the log-posterior at the mode to define the (full) covariance matrix. This procedure is computationally feasible only for a fully connected decoder model. We conduct experiments on MNIST and OMNIGLOT datasets where the fully connected decoder network consists of two hidden layers and the hidden layer dimensions are chosen to set the total number of weight parameters roughly equal to the convolutional decoder network used in the main paper.

Tab. 9 summarizes the results. Among the fully connected networks, the VLAE achieves the highest performance. Instead of doing SVI gradient updates as in the SAVI method (SA), the VLAE aims to directly solve for the mode of the true posterior by decoder linearization, leading to more accurate posterior refinement without suffering from the step size issue. Our recursive mixture, with the fully connected decoder networks, still improves the VAE's scores, but the improvement is often less than that of the VLAE. However, when compared to the convnet decoder cases, even the conventional VAE significantly outperforms the VLAE. The best VLAE's scores are significantly lower than VAE's using convolutional decoders. Restricted network architecture of the VLAE is its main drawback.

We also compare the test inference times of our recursive mixture model and the VLAE using the fully connected decoder networks. Note that VLAE is a semi-amortized approach, which needs to solve the Laplace approximation at test time. Thus another drawback of VLAE is the computational overhead of inference, which can be demanding as the number of linearization steps increases. The per-batch inference times (batch size 128) are shown in Tab. 10. For the moderate or large linearization steps (e.g., 4 or 8), the inference takes significantly longer than that of our RME (amortized method).

Table 10: (Fully connected networks as decoders) Per-batch inference time (unit in milliseconds) with batch size 128. The figures without parentheses are the times using the fully connected networks, whereas figures in the parentheses are the times using the convolutional decoder networks.

	MNIST		OMNIGLOT	
	$\text{DIM}(\mathbf{z}) = 10$	$\text{DIM}(\mathbf{z}) = 50$	$\text{DIM}(\mathbf{z}) = 10$	$\text{DIM}(\mathbf{z}) = 50$
VLAE (1)	10.1	12.9	11.2	12.1
VLAE (2)	11.2	13.4	13.2	16.9
VLAE (4)	14.8	17.8	15.4	18.7
VLAE (8)	20.7	30.8	22.1	26.4
RME (2)	5.0 (5.0)	5.0 (4.7)	5.4 (6.0)	5.6 (5.4)
RME (3)	4.9 (5.1)	4.9 (4.9)	5.9 (5.7)	5.4 (5.5)
RME (4)	4.9 (5.0)	4.9 (4.6)	6.1 (5.9)	5.9 (5.3)
RME (5)	5.0 (5.1)	4.7 (4.8)	5.8 (6.1)	5.4 (5.6)

10 Pseudo Codes

The following is the pseudocode for the proposed model. The real full Python/PyTorch code is available in `https://github.com/minyoungkim21/recmixvae`.

```
#### Hyperparameters ####

batch_size = 128                # input batch size for training
n_epochs = 2000                 # number of epochs to train
x_dim = (C=1 x H=28 x W=28)     # input dimension
z_dim = 50                      # latent space dimension
learning_rate = 1e-6            # learning rate for ADAM optimizer

num_comps = 5                   # number of mixture components for encoder
eps_regr_nhl = 1                # number of hidden layers for epsilon regressor
eps_regr_dim = 10               # hidden layer dim for epsilon regressor
eps_min = 0.001                 # minimum epsilon
eps_max = 0.1                   # maximum epsilon
kl_max = 500.0                  # maximum kl(q_k||Q_{k-1}) allowed in the objective


#### Main class ####

import torch.nn as nn

class RecMixVAE(nn.Module):

    self.M = num_comps-1  # components: 0,1,...,M (the number of comps = M+1)
    self.decoder = ConvDecoder(z_dim, x_dim)  # decoder
    self.prior = DiagonalGaussian(mu=zeros, logvar=zeros)  # prior

    # components of encoder (q_0, q_1, ..., q_M)
    self.comps = nn.ModuleList( [ConvEncoder(z_dim, x_dim) for _ in range(num_comps)] )

    # regressors for impacts of components  (eps_0, eps_1, ..., eps_M); note: eps_0 = 1 (const)
    self.eps_regrs = nn.ModuleList( [Const(1.0)] +
        [ BaseBoundedRegressor( x_dim, eps_min, eps_max, eps_regr_nhl, eps_regr_dim )
          for _ in range(num_comps-1) ] )

    def encoder_upto_kth(self, x, k):
        '''
        Mixture with components q_0(.|x), q_1(.|x), ..., q_k(.|x) is formed.
        More specifically, eg, for k=2,
          Q_{k=2}(.|x) = alpha_0(x) * q_0(.|x) + alpha_1(x) * q_1(.|x) + alpha_2(x) * q_2(.|x)
        where
          alpha_2(x) = eps_2(x)
          alpha_1(x) = eps_1(x) * (1-eps_2(x))
          alpha_0(x) = eps_0(x) * (1-eps_1(x)) * (1-eps_2(x))
        inputs:
          k = component index (0 <= k <= self.M)
        returns:
          n mixtures for Q_k(.|x) (with k+1 components)
        ’’’

    def encoder_kth_comp(self, x, k):
        ’’’
        Just return k-th component q_k(.|x)
        inputs:
          k = component index (0 <= k <= self.M)
        returns:
          n distributions (eg, DiagonalGaussian’s) q_k(.|x)
        ’’’
        return self.comps[k](x)[0]

    def eval_elbo_for_mixture(self, x, mixture):
        ’’’
        Evaluate elbo (recon error and kl) for a mixture encoder
        inputs:
          mixture = n mixture distributions from Q(.|x)
        returns:
          ell = E_{Q(z|x)}[ log p(x|z) ]
          kl = KL( Q(z|x) || p(z) )
        ’’’
        let K = mixture order
        alphas = mixture.logalphas.exp()
        z = samples from q_m(z|x) for m=1...K
        (decoder) evaluate log p(x|z) for z ~ q_m(z|x) for m=1...K
        (prior) evaluate log p(z) for z ~ q_m(z|x) for m=1...K
        evaluate log Q(z|x) for z ~ q_m(z|x) for m=1...K
        return ell = E_{Q(z|x)}[ log p(x|z) ] and kl = KL( Q(z|x) || p(z) )

    def forward(self, x, k, loss_type):
        ’’’
        compute objectives for recursive mixture VAE
        inputs:
          k = component index (0 <= k <= self.M)
          loss_type = either of
              ’new_comp’: compute elbo(q_k) and kl(q_k||Q_{k-1}) (the latter None if k=0)
              ’mixture’: compute elbo(Q_k)
        returns:
          loss_type == ’new_comp’: elbo(q_k), kl(q_k||Q_{k-1}) (averaged over batch x)
          loss_type == ’mixture’: elbo(Q_k) (averaged over batch x)
        ’’’
      if loss_type == ’new_comp’:
          q_z_x = self.encoder_kth_comp(x, k)  # q_k
          Q_z_x = self.encoder_upto_kth(x, k-1) if k>0 else None  # Q_{k-1}
          evaluate elbo(q_k) and kl(q_k||Q_{k-1})
      elif loss_type == ’mixture’:
          Q_z_x = self.encoder_upto_kth(x, k)  # Q_k
          ell, kl = self.eval_elbo_for_mixture(x, Q_z_x)
          elbo = ( ell - kl ).mean()

    def enable_grad(self, params):
        ’’’
        Disable the autograd for all parameters except for "params"
        ’’’


#### Main algorithm ####

model = RecMixVAE()

while epoch <= n_epochs:
```

```
for batch sampled from the training data:

    # update q_0
    model.enable_grad(model.comps[0])
    elbo, _ = model(batch, 0, loss_type='new_comp')
    update model by backprop with loss = -elbo

    # update (q_m, eps_regr_m) for m=1,...,M
    for m in range(1,model.M+1):

        # update q_m
        model.enable_grad(model.comps[m])
        elbo, kl = model(batch, m, loss_type='new_comp')
        update model by backprop with loss = -elbo + (kl_max - kl).relu()

        # update eps_regr_m
        model.enable_grad(model.eps_regrs[m])
        elbo = model(batch, m, loss_type='mixture')
        update model by backprop with loss = -elbo

    # update decoder
    model.enable_grad(model.decoder)
    elbo = model(batch, model.M, loss_type='mixture')
    update model by backprop with loss = -elbo
```

Table 11: (MNIST) Test log-likelihood scores (unit in nat) estimated by the importance weighted sampling [1]. The figures in the parentheses next to model names indicate: the number of SVI steps in SA, the number of flows in IAF and HF, and the number of mixture components in ME and RME. The superscripts are the standard deviations. The best (on average) results are boldfaced in **red**. In each column, the statistical significance of the difference between the best model (red) and each competing model, is depicted as color: anything non-colored indicates $p \leq 0.01$ (strongly distinguished), $p \in (0.01, 0.05]$ as yellow-orange, $p \in (0.05, 0.1]$ as orange, $p > 0.1$ as red orange (little evidence of difference) by the Wilcoxon signed rank test. Best viewed in color.

$\dim(\mathbf{z})$	10	20	50	100
VAE	$685.1^{1.8}$	$930.7^{3.9}$	$1185.7^{3.9}$	$1225.4^{4.2}$
SA$^{(1)}$	$688.1^{2.7}$	$921.2^{2.3}$	$1172.1^{1.8}$	$1196.9^{3.3}$
SA$^{(2)}$	$682.2^{1.5}$	$932.0^{2.4}$	$1176.3^{3.4}$	$1216.7^{2.9}$
SA$^{(4)}$	$683.5^{1.5}$	$925.5^{2.6}$	$1171.3^{3.5}$	$1217.7^{3.9}$
SA$^{(8)}$	$684.6^{1.5}$	$928.1^{3.9}$	$1183.2^{3.4}$	$1211.7^{2.9}$
IAF$^{(1)}$	$687.3^{1.1}$	$934.0^{3.3}$	$1180.6^{2.7}$	$1213.4^{5.6}$
IAF$^{(2)}$	$677.7^{1.6}$	$931.4^{3.7}$	$1190.1^{1.9}$	$1224.4^{2.2}$
IAF$^{(4)}$	$685.0^{1.5}$	$926.3^{2.6}$	$1178.1^{1.6}$	$1216.4^{3.9}$
IAF$^{(8)}$	$689.7^{1.4}$	$934.1^{2.4}$	$1150.0^{2.2}$	$1190.9^{3.9}$
HF$^{(1)}$	$682.5^{1.4}$	$917.2^{2.6}$	$1204.3^{4.0}$	$1203.3^{2.3}$
HF$^{(2)}$	$677.6^{2.2}$	$923.9^{3.1}$	$1191.5^{10.8}$	$1213.6^{3.0}$
HF$^{(4)}$	$683.3^{2.6}$	$927.3^{2.8}$	$1197.2^{1.5}$	$1226.0^{2.0}$
HF$^{(8)}$	$679.6^{1.5}$	$928.5^{3.1}$	$1184.1^{1.8}$	$1220.0^{3.5}$
ME$^{(2)}$	$685.7^{1.2}$	$926.7^{3.0}$	$1152.8^{1.7}$	$1191.4^{2.5}$
ME$^{(3)}$	$678.5^{2.5}$	$933.1^{4.1}$	$1162.8^{4.7}$	$1216.9^{2.1}$
ME$^{(4)}$	$680.0^{0.9}$	$914.7^{2.3}$	$\mathbf{1205.1}^{2.3}$	$1214.9^{3.4}$
ME$^{(5)}$	$682.0^{1.7}$	$920.6^{1.9}$	$1198.5^{3.5}$	$1181.7^{3.7}$
RME$^{(2)}$	$697.2^{1.1}$	$943.9^{1.6}$	$1201.7^{0.9}$	$1240.7^{2.5}$
RME$^{(3)}$	$698.2^{1.1}$	$945.1^{1.6}$	$1202.4^{1.0}$	$1240.8^{2.4}$
RME$^{(4)}$	$699.0^{1.0}$	$\mathbf{945.2}^{1.6}$	$1203.1^{1.0}$	$1241.5^{2.4}$
RME$^{(5)}$	$\mathbf{699.4}^{2.1}$	$945.0^{1.7}$	$1203.7^{1.0}$	$\mathbf{1242.0}^{2.4}$
BVI-ER1$^{(2)}$	$694.5^{1.9}$	$939.7^{2.8}$	$1196.2^{2.8}$	$1236.3^{3.0}$
BVI-ER1$^{(3)}$	$694.5^{1.9}$	$939.5^{2.9}$	$1191.6^{2.9}$	$1233.9^{3.0}$
BVI-ER1$^{(4)}$	$692.2^{1.8}$	$937.8^{2.9}$	$1191.6^{2.8}$	$1227.6^{3.0}$
BVI-ER1$^{(5)}$	$692.0^{1.9}$	$931.2^{3.0}$	$1183.1^{2.9}$	$1229.0^{3.1}$
BVI-ER2$^{(2)}$	$694.5^{1.9}$	$939.7^{2.1}$	$1189.6^{2.2}$	$1236.2^{3.0}$
BVI-ER2$^{(3)}$	$694.5^{1.9}$	$939.4^{2.1}$	$1192.1^{2.3}$	$1233.6^{3.0}$
BVI-ER2$^{(4)}$	$692.2^{1.9}$	$937.6^{2.1}$	$1191.5^{2.2}$	$1227.4^{3.0}$
BVI-ER2$^{(5)}$	$692.4^{1.9}$	$931.7^{2.2}$	$1181.7^{2.2}$	$1228.9^{3.0}$

Table 12: (OMNIGLOT) Test log-likelihood scores (unit in nat). The same interpretation as Tab. 11.

dim($\mathbf{z}$)	10	20	50	100
VAE	$347.0^{1.7}$	$501.6^{1.6}$	$801.6^{4.0}$	$917.5^{5.1}$
SA$^{(1)}$	$344.1^{1.4}$	$499.3^{2.5}$	$792.7^{7.9}$	$905.8^{4.2}$
SA$^{(2)}$	$349.5^{1.4}$	$501.0^{2.7}$	$793.1^{4.8}$	$920.0^{4.5}$
SA$^{(4)}$	$342.1^{1.0}$	$488.2^{1.8}$	$794.4^{1.9}$	$914.6^{5.6}$
SA$^{(8)}$	$344.8^{1.1}$	$490.3^{2.8}$	$799.4^{2.7}$	$942.2^{5.2}$
IAF$^{(1)}$	$347.8^{1.6}$	$489.9^{1.9}$	$788.8^{4.1}$	$937.4^{7.2}$
IAF$^{(2)}$	$344.2^{1.6}$	$494.9^{1.4}$	$795.7^{2.7}$	$934.6^{7.3}$
IAF$^{(4)}$	$347.9^{1.9}$	$496.0^{2.0}$	$775.1^{2.2}$	$920.9^{4.1}$
IAF$^{(8)}$	$343.9^{1.4}$	$498.8^{2.3}$	$774.7^{2.9}$	$885.7^{2.8}$
HF$^{(1)}$	$335.5^{1.2}$	$488.6^{2.0}$	$795.9^{3.3}$	$917.0^{2.4}$
HF$^{(2)}$	$340.6^{1.3}$	$495.9^{1.8}$	$784.5^{4.8}$	$929.4^{3.7}$
HF$^{(4)}$	$343.3^{1.2}$	$487.0^{2.7}$	$799.7^{3.2}$	$877.5^{4.7}$
HF$^{(8)}$	$343.3^{1.3}$	$488.3^{2.4}$	$794.6^{4.0}$	$889.2^{4.7}$
ME$^{(2)}$	$344.2^{1.5}$	$491.7^{1.4}$	$793.4^{3.8}$	$880.3^{3.6}$
ME$^{(3)}$	$350.3^{1.8}$	$491.2^{2.1}$	$807.5^{4.9}$	$875.9^{4.6}$
ME$^{(4)}$	$337.7^{1.1}$	$491.3^{1.8}$	$732.0^{3.1}$	$939.8^{8.6}$
ME$^{(5)}$	$343.0^{1.4}$	$478.0^{2.8}$	$805.7^{3.8}$	$861.9^{7.0}$
RME$^{(2)}$	$349.3^{1.5}$	$508.2^{1.2}$	$\mathbf{821.0}^{3.1}$	$941.5^{1.7}$
RME$^{(3)}$	$349.9^{1.6}$	$507.5^{1.1}$	$820.4^{0.9}$	$\mathbf{944.6}^{5.1}$
RME$^{(4)}$	$350.7^{1.7}$	$509.0^{1.2}$	$819.9^{0.9}$	$944.4^{1.7}$
RME$^{(5)}$	$\mathbf{351.1}^{1.7}$	$\mathbf{509.1}^{1.4}$	$819.9^{0.9}$	$944.0^{1.6}$
BVI-ER1$^{(2)}$	$349.2^{1.9}$	$507.9^{2.2}$	$817.1^{3.3}$	$937.9^{5.1}$
BVI-ER1$^{(3)}$	$350.0^{1.9}$	$507.8^{2.2}$	$816.6^{3.4}$	$936.2^{5.1}$
BVI-ER1$^{(4)}$	$350.7^{1.5}$	$507.8^{2.3}$	$816.8^{3.4}$	$935.6^{3.8}$
BVI-ER1$^{(5)}$	$351.1^{1.5}$	$508.2^{2.3}$	$816.4^{3.3}$	$935.7^{3.8}$
BVI-ER2$^{(2)}$	$349.3^{1.9}$	$507.8^{2.2}$	$817.1^{3.4}$	$937.6^{5.1}$
BVI-ER2$^{(3)}$	$349.8^{1.9}$	$507.8^{2.2}$	$816.6^{3.4}$	$936.1^{5.1}$
BVI-ER2$^{(4)}$	$350.7^{1.5}$	$507.8^{2.2}$	$816.9^{3.4}$	$935.6^{3.8}$
BVI-ER2$^{(5)}$	$351.0^{1.5}$	$508.1^{2.2}$	$816.4^{3.4}$	$935.7^{3.8}$

Table 13: (CIFAR10) Test log-likelihood scores (unit in nat). The same interpretation as Tab. 11.

dim($\mathbf{z}$)	10	20	50	100
VAE	$1645.7^{4.9}$	$2089.7^{5.8}$	$2769.9^{7.1}$	$3381.0^{14.7}$
SA$^{(1)}$	$1645.0^{5.6}$	$2086.0^{6.2}$	$2765.0^{7.1}$	$3378.7^{10.4}$
SA$^{(2)}$	$1648.6^{4.8}$	$2088.2^{6.6}$	$2764.1^{7.7}$	$3377.8^{9.8}$
SA$^{(4)}$	$1648.5^{5.2}$	$2083.9^{8.4}$	$2766.7^{6.6}$	$3380.2^{7.9}$
SA$^{(8)}$	$1642.1^{5.4}$	$2086.0^{6.1}$	$2766.6^{7.5}$	$3376.6^{10.6}$
IAF$^{(1)}$	$1646.0^{4.9}$	$2081.1^{5.4}$	$2762.6^{7.2}$	$3383.7^{7.1}$
IAF$^{(2)}$	$1642.0^{4.9}$	$2084.6^{5.6}$	$2763.0^{4.3}$	$3373.3^{14.2}$
IAF$^{(4)}$	$1646.0^{5.1}$	$2083.2^{6.1}$	$2760.6^{7.0}$	$3371.1^{8.1}$
IAF$^{(8)}$	$1643.6^{4.6}$	$2087.1^{4.6}$	$2761.8^{6.9}$	$3364.0^{9.6}$
HF$^{(1)}$	$1644.5^{4.4}$	$2079.1^{5.5}$	$2757.9^{4.4}$	$3393.4^{4.7}$
HF$^{(2)}$	$1636.7^{4.9}$	$2086.0^{5.9}$	$2764.7^{4.4}$	$3384.8^{4.7}$
HF$^{(4)}$	$1642.1^{4.9}$	$2082.3^{7.3}$	$2763.4^{4.4}$	$3385.5^{4.4}$
HF$^{(8)}$	$1639.9^{5.4}$	$2084.7^{6.1}$	$2765.5^{7.2}$	$3382.5^{4.3}$
ME$^{(2)}$	$1643.6^{5.1}$	$2086.6^{6.8}$	$2767.9^{9.4}$	$3378.5^{9.1}$
ME$^{(3)}$	$1638.6^{5.8}$	$2079.8^{5.9}$	$2770.2^{7.8}$	$3388.1^{7.7}$
ME$^{(4)}$	$1641.8^{5.4}$	$2084.7^{6.9}$	$2763.5^{9.3}$	$3384.6^{10.3}$
ME$^{(5)}$	$1641.7^{5.6}$	$2080.2^{5.9}$	$2766.1^{6.3}$	$3351.3^{11.0}$
RME$^{(2)}$	$1652.3^{5.0}$	$2095.7^{5.8}$	$2779.6^{6.6}$	$3403.0^{6.9}$
RME$^{(3)}$	$1654.2^{4.9}$	$\mathbf{2099.1}^{7.2}$	$\mathbf{2783.0}^{6.1}$	$3404.2^{6.8}$
RME$^{(4)}$	$\mathbf{1655.0}^{6.4}$	$2096.6^{5.9}$	$2781.1^{6.6}$	$3403.2^{6.1}$
RME$^{(5)}$	$1654.5^{4.6}$	$2098.4^{5.8}$	$2782.9^{6.4}$	$\mathbf{3404.6}^{5.7}$
BVI-ER1$^{(2)}$	$1648.6^{5.1}$	$2094.4^{5.7}$	$2775.9^{6.4}$	$3393.1^{6.8}$
BVI-ER1$^{(3)}$	$1648.9^{5.0}$	$2094.7^{5.9}$	$2776.2^{6.6}$	$3393.8^{6.5}$
BVI-ER1$^{(4)}$	$1649.0^{5.1}$	$2095.0^{5.8}$	$2776.5^{6.3}$	$3394.2^{6.6}$
BVI-ER1$^{(5)}$	$1649.1^{5.2}$	$2095.1^{5.8}$	$2776.8^{6.5}$	$3394.2^{7.7}$
BVI-ER2$^{(2)}$	$1648.6^{5.1}$	$2094.4^{5.7}$	$2775.8^{6.8}$	$3393.1^{6.6}$
BVI-ER2$^{(3)}$	$1648.9^{5.0}$	$2094.7^{5.7}$	$2776.2^{6.6}$	$3393.8^{6.5}$
BVI-ER2$^{(4)}$	$1649.0^{5.1}$	$2095.0^{5.8}$	$2776.5^{6.3}$	$3394.2^{6.2}$
BVI-ER2$^{(5)}$	$1649.1^{5.1}$	$2095.1^{5.8}$	$2776.8^{6.5}$	$3394.1^{6.1}$

Table 14: (SVHN) Test log-likelihood scores (unit in nat). The same interpretation as Tab. 11.

dim($\mathbf{z}$)	10	20	50	100
VAE	$3360.2^{9.1}$	$4054.5^{14.3}$	$5363.7^{21.4}$	$6703.0^{28.4}$
SA$^{(1)}$	$3358.7^{8.9}$	$4031.5^{19.0}$	$5362.1^{35.7}$	$6707.6^{24.8}$
SA$^{(2)}$	$3356.0^{8.8}$	$4041.5^{15.5}$	$5377.0^{23.2}$	$6697.0^{35.5}$
SA$^{(4)}$	$3327.8^{8.2}$	$4051.9^{22.2}$	$5391.7^{20.4}$	$6645.1^{19.8}$
SA$^{(8)}$	$3352.8^{11.5}$	$4041.6^{9.5}$	$5370.8^{18.5}$	$6674.5^{20.9}$
IAF$^{(1)}$	$3377.1^{8.4}$	$4050.0^{9.4}$	$5368.3^{11.5}$	$6650.3^{15.7}$
IAF$^{(2)}$	$3362.3^{8.9}$	$4054.6^{10.5}$	$5360.0^{10.0}$	$6671.5^{16.8}$
IAF$^{(4)}$	$3346.1^{8.7}$	$4048.6^{8.7}$	$5338.1^{10.2}$	$6630.0^{17.2}$
IAF$^{(8)}$	$3372.6^{8.3}$	$4042.0^{9.6}$	$5341.8^{10.1}$	$6602.0^{10.8}$
HF$^{(1)}$	$3381.4^{8.9}$	$4028.8^{9.7}$	$5372.0^{10.1}$	$6678.8^{8.8}$
HF$^{(2)}$	$3342.4^{8.3}$	$4030.7^{9.9}$	$5376.6^{10.2}$	$6672.0^{9.6}$
HF$^{(4)}$	$3370.0^{8.2}$	$4038.4^{9.7}$	$5371.8^{9.8}$	$6655.2^{9.5}$
HF$^{(8)}$	$3343.8^{8.2}$	$4035.9^{8.9}$	$5351.1^{11.1}$	$6642.4^{16.5}$
ME$^{(2)}$	$3352.3^{9.9}$	$4037.2^{11.0}$	$5343.2^{13.1}$	$6670.2^{46.5}$
ME$^{(3)}$	$3335.2^{10.9}$	$4053.8^{16.1}$	$5367.7^{15.8}$	$6605.6^{9.4}$
ME$^{(4)}$	$3358.2^{14.9}$	$4061.3^{12.0}$	$5191.9^{18.5}$	$6605.7^{9.2}$
ME$^{(5)}$	$3360.6^{7.8}$	$4057.5^{12.2}$	$5209.2^{12.8}$	$6604.0^{16.6}$
RME$^{(2)}$	$3390.0^{8.1}$	$4085.3^{9.7}$	$5403.2^{10.2}$	$\mathbf{6784.7}^{25.0}$
RME$^{(3)}$	$\mathbf{3392.0}^{12.6}$	$4085.9^{9.8}$	$5405.1^{10.4}$	$6782.7^{9.3}$
RME$^{(4)}$	$3388.6^{8.3}$	$4080.7^{9.9}$	$5403.8^{10.2}$	$6780.2^{9.4}$
RME$^{(5)}$	$3391.9^{8.2}$	$\mathbf{4086.9}^{10.9}$	$\mathbf{5405.5}^{8.5}$	$6781.8^{10.0}$
BVI-ER1$^{(2)}$	$3379.9^{8.2}$	$4077.3^{10.3}$	$5388.2^{10.2}$	$6753.5^{10.0}$
BVI-ER1$^{(3)}$	$3380.9^{8.1}$	$4076.6^{10.3}$	$5384.2^{10.5}$	$6750.3^{10.6}$
BVI-ER1$^{(4)}$	$3384.4^{8.1}$	$4073.1^{10.2}$	$5371.1^{10.4}$	$6748.9^{11.3}$
BVI-ER1$^{(5)}$	$3382.2^{8.4}$	$4071.2^{10.2}$	$5378.1^{10.1}$	$6733.6^{15.3}$
BVI-ER2$^{(2)}$	$3379.8^{8.1}$	$4077.3^{9.8}$	$5388.3^{10.1}$	$6753.2^{10.1}$
BVI-ER2$^{(3)}$	$3380.9^{8.4}$	$4076.7^{9.6}$	$5383.9^{10.2}$	$6749.7^{10.7}$
BVI-ER2$^{(4)}$	$3384.3^{8.2}$	$4073.2^{9.2}$	$5371.3^{10.4}$	$6749.1^{11.1}$
BVI-ER2$^{(5)}$	$3382.1^{8.4}$	$4071.2^{10.4}$	$5377.7^{10.2}$	$6733.8^{15.0}$

Table 15: (CelebA) Test log-likelihood scores (unit in nat). The same interpretation as Tab. 11.

$\dim(\mathbf{z})$	10	20	50	100
VAE	$9767.7^{36.0}$	$12116.4^{25.3}$	$15251.9^{39.7}$	$17395.5^{32.4}$
SA$^{(1)}$	$9735.2^{21.4}$	$12091.1^{21.6}$	$15285.8^{29.4}$	$17432.4^{30.4}$
SA$^{(2)}$	$9754.2^{20.4}$	$12087.1^{21.5}$	$15252.7^{29.0}$	$17434.0^{29.8}$
SA$^{(4)}$	$9769.1^{20.6}$	$12116.3^{20.5}$	$15187.3^{27.9}$	$17360.5^{28.9}$
SA$^{(8)}$	$9744.8^{19.4}$	$12100.6^{22.8}$	$15096.5^{27.2}$	$17409.7^{28.0}$
IAF$^{(1)}$	$9750.3^{27.4}$	$12098.0^{20.6}$	$15271.2^{28.6}$	$17446.4^{30.3}$
IAF$^{(2)}$	$9794.4^{23.3}$	$12104.5^{21.8}$	$15262.2^{27.8}$	$17449.5^{31.8}$
IAF$^{(4)}$	$9764.7^{29.5}$	$12094.6^{22.6}$	$15261.0^{28.1}$	$17416.8^{29.8}$
IAF$^{(8)}$	$9764.0^{21.6}$	$12109.3^{22.0}$	$15241.5^{27.9}$	$17452.5^{39.5}$
HF$^{(1)}$	$9748.3^{29.5}$	$12077.2^{31.4}$	$15240.5^{27.6}$	$17461.6^{29.9}$
HF$^{(2)}$	$9765.8^{25.6}$	$12093.0^{25.6}$	$15258.2^{30.3}$	$17479.8^{30.0}$
HF$^{(4)}$	$9754.3^{23.8}$	$12082.0^{27.0}$	$15266.5^{29.5}$	$17532.7^{30.6}$
HF$^{(8)}$	$9737.5^{24.5}$	$12087.3^{25.5}$	$15248.7^{29.7}$	$17663.4^{28.7}$
ME$^{(2)}$	$9825.3^{20.7}$	$12072.7^{23.3}$	$15290.5^{29.3}$	$17419.3^{28.7}$
ME$^{(3)}$	$9797.6^{22.3}$	$12100.3^{21.7}$	$15294.6^{28.3}$	$17395.3^{28.9}$
ME$^{(4)}$	$9834.9^{25.4}$	$12092.2^{22.6}$	$15270.7^{20.6}$	$17458.5^{36.8}$
ME$^{(5)}$	$9717.0^{23.2}$	$12095.3^{25.1}$	$15268.8^{27.5}$	$17406.8^{31.8}$
RME$^{(2)}$	$9837.9^{24.6}$	$12193.1^{23.5}$	$15363.0^{31.7}$	$17873.5^{32.8}$
RME$^{(3)}$	$9838.5^{25.0}$	$12192.3^{23.5}$	$15365.6^{31.4}$	$17874.4^{31.2}$
RME$^{(4)}$	$\mathbf{9849.5}^{12.1}$	$12192.6^{23.4}$	$15364.3^{31.5}$	$\mathbf{17875.1}^{14.2}$
RME$^{(5)}$	$9843.5^{25.0}$	$\mathbf{12194.2}^{11.5}$	$\mathbf{15366.2}^{12.7}$	$17874.3^{32.5}$
BVI-ER1$^{(2)}$	$9801.6^{26.1}$	$12133.5^{25.1}$	$15206.4^{28.2}$	$17716.9^{70.3}$
BVI-ER1$^{(3)}$	$9805.6^{25.7}$	$12146.5^{22.4}$	$15249.5^{28.1}$	$17558.6^{120.1}$
BVI-ER1$^{(4)}$	$9805.2^{29.3}$	$12127.7^{22.3}$	$15085.8^{28.4}$	$17256.1^{283.9}$
BVI-ER1$^{(5)}$	$9810.1^{30.7}$	$12092.3^{22.3}$	$15052.5^{28.0}$	$17069.9^{391.8}$
BVI-ER2$^{(2)}$	$9801.5^{25.3}$	$12133.6^{28.7}$	$15207.3^{52.4}$	$17716.6^{92.1}$
BVI-ER2$^{(3)}$	$9805.7^{24.9}$	$12146.6^{25.5}$	$15249.6^{54.6}$	$17560.7^{109.2}$
BVI-ER2$^{(4)}$	$9805.1^{26.3}$	$12128.7^{34.0}$	$15084.9^{42.5}$	$17260.6^{228.6}$
BVI-ER2$^{(5)}$	$9810.4^{27.8}$	$12087.5^{48.9}$	$15051.7^{43.5}$	$17077.1^{387.6}$

References

[1] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders, 2016. In Proceedings of the Second International Conference on Learning Representations, ICLR.

[2] Trevor Campbell and Xinglong Li. Universal boosting variational inference, 2019. In Advances in Neural Information Processing Systems.

[3] Casey Chu, Jose Blanchet, and Peter Glynn. Probability functional descent: A unifying perspective on GANs, variational inference, and reinforcement learning, 2019. International Conference on Machine Learning.

[4] Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning*, 2018.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

[6] J. Friedman. Greedy function approximation: A gradient boosting machine, 1999. Technical Report, Dept. of Statistics, Stanford University.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets, 2014. In Advances in Neural Information Processing Systems.

[8] Fangjian Guo, Xiangyu Wang, Kai Fan, Tamara Broderick, and David B Dunson. Boosting variational inference. In *arXiv preprint*, 2016.

[9] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 13:1303–1347, 2013.

[10] Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization, 2013. International Conference on Machine Learning.

[11] Y. Kim, S. Wiseman, A. C. Millter, D. Sontag, and A. M. Rush. Semi-amortized variational autoencoders. In *International Conference on Machine Learning*, 2018.

[12] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow, 2016. In Advances in Neural Information Processing Systems.

[13] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes, 2014. In Proceedings of the Second International Conference on Learning Representations, ICLR.

[14] R. G. Krishnan, D. Liang, and M. D. Hoffman. On the challenges of learning with inference networks on sparse high-dimensional data. In *Artificial Intelligence and Statistics*, 2018.

[15] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009. Technical report, Computer Science Department, University of Toronto.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks, 2012. In Advances in Neural Information Processing Systems.

[17] Anna Kuzina, Evgenii Egorov, and Evgeny Burnaev. Boovae: A scalable framework for continual VAE learning under boosting approach. In *arXiv preprint*, 2019.

[18] B. M. Lake, R. R. Salakhutdinov, and J. Tenenbaum. One-shot learning by inverting a compositional causal process, 2013. In Advances in Neural Information Processing Systems.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[21] Francesco Locatello, Gideon Dresdner, Rajiv Khanna, Isabel Valera, and Gunnar Rätsch. Boosting black box variational inference, 2018. In Advances in Neural Information Processing Systems.

[22] Francesco Locatello, Rajiv Khanna, Joydeep Ghosh, and Gunnar Rätsch. Boosting variational inference: an optimization perspective, 2018. AI and Statistics (AISTATS).

[23] J. Marino, Y. Yisong, and S. Mandt. Iterative amortized inference. In *International Conference on Machine Learning*, 2018.

[24] L. Mason, J. Baxter, P. Bartlett, and M. Frean. *Functional gradient techniques for combining hypotheses*. In Advances in Large Margin Classifiers, MIT Press, 1999.

[25] Andrew C. Miller, Nicholas J. Foti, and Ryan P. Adams. Variational boosting: Iteratively refining posterior approximations, 2017. International Conference on Machine Learning.

[26] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[27] Yookoon Park, Chris Kim, and Gunhee Kim. Variational Laplace autoencoders. In *International Conference on Machine Learning*, 2019.

[28] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *arXiv preprint*, 2015.

[29] D.J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models, 2014. International Conference on Machine Learning.

[30] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *arXiv preprint*, 2013.

[31] J. M. Tomczak and M. Welling. Improving variational autoencoders using Householder flow, 2016. In Advances in Neural Information Processing Systems, Workshop on Bayesian Deep Learning.

[32] Jakub M. Tomczak and Max Welling. VAE with a VampPrior, 2018. Artificial Intelligence and Statistics.

[33] Mingzhang Yin and Mingyuan Zhou. Semi-implicit variational inference, 2018. International Conference on Machine Learning.

[34] O. Zobay. Variational bayesian inference with gaussian-mixture approximations. *Electron. J. Statist.*, 8(1):335–389, 2014.