SHALLOW VAES WITH REALNVP PRIOR CAN PERFORM AS WELL AS DEEP HIERARCHICAL VAES

Anonymous authors
Paper under double-blind review

ABSTRACT

Using powerful posterior distributions is a popular technique in variational inference. However, recent works showed that the aggregated posterior may fail to match unit Gaussian prior, even with expressive posteriors, thus learning the prior becomes an alternative way to improve the variational lower-bound. We show that using learned RealNVP prior and just one latent variable in VAE, we can achieve test NLL comparable to very deep state-of-the-art hierarchical VAE, outperforming many previous works with complex hierarchical VAE architectures. We hypothesize that, when coupled with Gaussian posteriors, the learned prior can encourage appropriate posterior overlapping, which is likely to improve reconstruction loss and lower-bound, supported by our experimental results. We demonstrate that, with learned RealNVP prior, $\beta$-VAE can have better rate-distortion curve than using fixed Gaussian prior.

1 INTRODUCTION

Variational auto-encoder (VAE) (Kingma & Welling, 2014; Rezende et al., 2014) is a powerful deep generative model. The use of amortized variational inference makes VAE scalable to deep neural networks and large amount of data. Variational inference demands the intractable true posterior to be approximated by a tractable distribution. The original VAE used factorized Gaussian for both the prior and the variational posterior (Kingma & Welling, 2014; Rezende et al., 2014). Since then, lots of more expressive variational posteriors have been proposed (Tran et al., 2016; Rezende & Mohamed, 2015; Salimans et al., 2015; Nalisnick et al., 2016; Kingma et al., 2016; Mescheder et al., 2017; van den Berg et al., 2018). However, recent work suggested that even with powerful posteriors, VAE may still fail to match aggregated posterior to unit Gaussian prior (Rosca et al., 2018), indicating there is still a gap between the approximated and the true posterior.

To improve the variational lower-bound, one alternative way to using powerful posterior distributions is to learn the prior, an idea initially suggested by Hoffman & Johnson (2016). Later on, Huang et al. (2017) applied RealNVP (Dinh et al., 2017) to learn the prior. Tomczak & Welling (2018) proved the optimal prior is the aggregated posterior, which they approximate by assembling a mixture of the posteriors with a set of learned pseudo-inputs. Bauer & Mnih (2019) constructed a rich prior by multiplying a simple prior with a learned acceptance function. Takahashi et al. (2019) introduced the kernel density trick to estimate the KL divergence in ELBO, for their implicit prior.

Despite the achievements of these previous works on posteriors and priors, the state-of-the-art VAE models with continuous latent variables all rely on deep hierarchical latent variables\footnote{The term “hierarchical latent variables” refers to multiple layers of latent variables, formulated as $p(x) = E_{q(z_1)} E_{q(z_2 | z_1)} \ldots E_{q(z_K | z_{K-1}, \ldots, z_1)} [p(x | z_1, \ldots, z_K)]$; while “one latent variable” refers to just one $z$ in standard VAEs. Both $z_0$ and $z$ are multi-dimensional tensors. Also, “deep” refers to many hierarchical latent variables, while “shallow” refers to few latent variables.}, although some of them might have used complicated posteriors/priors as components in their architectures. Most latent variables in such deep hierarchical VAES have no clear semantic meanings, just a technique for reaching good lower-bounds. This is in sharp contrast with GANs (Goodfellow et al., 2014; Arjovsky et al., 2017) and flow-based models (Ho et al., 2018), where most of them can reach state-of-the-art results with only one latent variable, with clear semantic meanings. We thus raise and answer a question: with the help of learned priors, can shallow VAEs achieve performance comparable or better than deep hierarchical VAEs? This question is important because a shallow
VAE would be much more promising to scale to more complicated datasets than deep hierarchical
VAEs. To answer this question, we conduct comprehensive experiments on several datasets with
learned RealNVP priors and just one latent variable, which even shows advantage over some deep
hierarchical VAEs with powerful posteriors. We also propose a hypothesis on why learning the prior
can lead to such great improvement in model performance, supported by our experimental results.
In summary, our contributions are:

- We conduct comprehensive experiments on four binarized datasets with four different net-
work architectures. Our results show that VAE with RealNVP prior consistently outper-
forms standard VAE and RealNVP posterior.
- We are the first to show that using learned RealNVP prior with just one latent variable
in VAE, it is possible to achieve test negative log-likelihoods (NLLs) comparable to very
deep state-of-the-art hierarchical VAE on these four datasets, outperforming many previous
works using complex hierarchical VAE equipped with rich priors/posteriors.
- We hypothesize that, when coupled with Gaussian posteriors, the learned prior can encour-
age appropriate posterior overlapping, which is likely to improve reconstruction loss and
lower-bound, supported by our experimental results.
- We demonstrate that, with learned RealNVP prior, \( \beta \)-VAE can have better rate-distortion
curve (Alemi et al., 2018) than using fixed Gaussian prior.

2 Preliminaries

2.1 Variational auto-encoder (VAE)

Variational auto-encoder (VAE) (Kingma & Welling, 2014; Rezende et al., 2014) is a deep probabilistic model. It uses a latent variable \( z \) with prior \( p_{\lambda}(z) \), and a conditional distribution \( p_{\theta}(x|z) \), to model the observed variable \( x \). \( p_{\theta}(x) \) is defined as \( \int p_{\theta}(x|z) p_{\lambda}(z) \, dz \), where \( p_{\theta}(x|z) \) is derived by a
neural network with parameter \( \theta \). \( \log p_{\theta}(x) \) is bounded below by evidence lower-bound (ELBO):

\[
\log p_{\theta}(x) \geq \mathcal{L}(x; \lambda, \theta, \phi) = \mathbb{E}_{q_{\phi}(x|z)} [\log p_{\theta}(x|z) + \log p_{\lambda}(z) - \log q_{\phi}(z|x)] \tag{1}
\]

where \( q_{\phi}(z|x) \) is the variational posterior to approximate \( p_{\theta}(z|x) \), derived by a neural network with parameter \( \phi \). In Eq. (2), the first term is the reconstruction loss of \( x \), and the second term is the
Kullback Leibler (KL) divergence between \( q_{\phi}(z|x) \) and \( p_{\lambda}(z) \). Optimizing \( q_{\phi}(z|x) \) and \( p_{\theta}(x|z) \)
with respect to empirical distribution \( p^*(x) \) can be achieved by maximizing the expected ELBO w.r.t. \( p^*(x) \):

\[
\mathcal{L}(\lambda, \theta, \phi) = \mathbb{E}_{p^*(x)} [\mathcal{L}(x; \lambda, \theta, \phi)] = \mathbb{E}_{p^*(x)} \mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x|z) + \log p_{\lambda}(z) - \log q_{\phi}(z|x)] \tag{3}
\]

A hyper-parameter \( \beta \) can be added to the training objective, in order to control the trade-off between
reconstruction loss and KL divergence, known as \( \beta \)-VAE (Higgins et al., 2017; Alemi et al., 2018):

\[
\mathcal{L}(\lambda, \theta, \phi) = \mathbb{E}_{p^*(x)} \mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x|z) + \beta (\log p_{\lambda}(z) - \log q_{\phi}(z|x))] \tag{4}
\]

The aggregated posterior \( q_{\phi}(z) = \int q_{\phi}(z|x) p^*(x) \, dx \), should be equal to \( p_{\lambda}(z) \), if VAE is perfectly trained (Hoffman & Johnson, 2016). However, even with powerful posteriors, the aggregated
posterior may still not match a unit Gaussian prior (Rosca et al., 2018). To investigate this problem,
Hoffman & Johnson (2016) suggested to decompose Eq. (3) as:

\[
\mathcal{L}(\lambda, \theta, \phi) = \mathbb{E}_{p^*(x)} \mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x|z)] - \operatorname{KL}(q_{\phi}(z) \| p_{\lambda}(z)) - \mathbb{H}[Z; X] \tag{5}
\]

where \( \mathbb{H}[Z; X] = \int \int q_{\phi}(z, x) \log \frac{q_{\phi}(z, x)}{q_{\phi}(z) p^*(x)} \, dz \, dx \) is the mutual information. Since \( p_{\lambda}(z) \) is only in \( \mathbb{H}[Z; X] \), ELBO can be further enlarged if \( p_{\lambda}(z) \) is trained to match \( q_{\phi}(z) \).

2.2 RealNVP

RealNVP (Dinh et al., 2017) is a deep probabilistic model. We denote its observed variable by \( z \) and latent variable by \( w \), with marginal distribution \( p_{\lambda}(z) \) and prior \( p_{\xi}(w) \). Unlike VAE, RealNVP
relates \( z \) and \( w \) by an invertible mapping \( w = f_\lambda(x) \) instead of a conditional distribution, as follows:

\[
p_\lambda(z) = p_\xi(w) \left| \det \left( \frac{\partial f_\lambda(z)}{\partial z} \right) \right| , \quad z = f_\lambda^{-1}(w)
\]

where \( \det \left( \frac{\partial f_\lambda(z)}{\partial z} \right) \) is the Jacobian determinant of \( f_\lambda \). In RealNVP, \( f_\lambda \) is composed of \( K \) invertible mappings, where \( f_\lambda(z) = (f_K \circ \cdots \circ f_1)(z) \), and each \( f_k \) is invertible. \( f_k \) must be carefully designed to ensure that the determinant can be computed efficiently. The original paper of RealNVP introduced the affine coupling layer as \( f_k \) [Kingma & Dhariwal, 2018] further introduced actnorm and invertible 1x1 convolution. Details can be found in their papers.

3 LEARNING THE PRIOR WITH REALNVP

It is straightforward to obtain a rich prior \( p_\lambda(z) \) from a simple (i.e., with constant parameters) one with RealNVP. Denote the simple prior as \( p_\xi(w) \), while the RealNVP mapping as \( w = f_\lambda(z) \). We then obtain Eq. (6) as our prior \( p_\lambda(z) \). Substitute Eq. (6) into (5), we get to the training objective:

\[
\mathcal{L}(\lambda, \theta, \phi) = E_{p^*(x)} E_{q_\phi(z|x)} \left[ \log p_\theta(x|z) + \log p_\xi(z) + \log \left| \det \left( \frac{\partial f_\lambda(z)}{\partial z} \right) \right| - \log q_\phi(z|x) \right]
\]

We mainly use joint training [Tomczak & Welling, 2018; Bauer & Mnih, 2019] (where \( p_\lambda(z) \) is jointly trained along with \( q_\phi(z|x) \) and \( p_\theta(x|z) \) by directly maximizing Eq. (7)), but we also consider other two strategies, see Appendices B.3 and B.6 for discussions.

In our initial experiments, training VAEs with deep RealNVP priors is unstable. The HVAEs (see Appendix B.10) with RealNVP priors are even more unstable, where the training procedure can barely finish. However, we tackle this problem by the following two techniques: (1) using gradient clip, as the original RealNVP (Dimi et al., 2017) does; and (2) clipping the std of Gaussian posterior \( q_\phi(z|x) \) of VAE (or \( q_\phi(z|x_0) \) and \( q_\phi(z_1|x) \) of HVAE) by a minimum value of \( e^{-1}\approx 1.67 \times 10^{-5} \).

4 APPROPRIATE POSTERIOR OVERLAPPING WITH LEARNED PRIOR

As we shall see in Section 5.4, RealNVP prior can result in a test negative log-likelihood (NLL) substantially better than RealNVP posterior, with the same flow depth. Also, using both RealNVP posterior and prior shows no significant advantage over using RealNVP prior only, although the total flow depth of the former variant is twice as large as the latter one. To investigate why VAE with RealNVP prior can lead to substantially better model performance than the other two variants, we take the particular architecture of our models into consideration (i.e., Bernoulli \( p_\theta(x|z) \), Gaussian \( q_\phi(z|x) \), and unit Gaussian or flow prior \( p_\lambda(z) \)), and start with the following proposition:

**Proposition 1.** Given a finite number of discrete training data, i.e., \( p^*(x) = \frac{1}{N} \sum_{i=1}^{N} \delta(x - x^{(i)}) \), if \( p_\theta(x|z) = \text{Bernoulli}(\mu_\theta(z)) \), where the Bernoulli mean \( \mu_\theta(z) \) is produced by the decoder, and \( 0 < \mu_\theta^2(z) < 1 \) for each of its \( k \)-dimensional output, then the optimal decoder \( \mu_\theta(z) \) is:

\[
\mu_\theta(z) = \sum_i w_i(z) x^{(i)}, \quad \text{where} \quad w_i(z) = \frac{q_\phi(z|x^{(i)})}{\sum_j q_\phi(z|x^{(j)})} \quad \text{and} \quad \sum_i w_i(z) = 1
\]

**Proof.** See Appendix A. \( \square \)

Proposition 1 suggests that if \( q_\phi(z|x) \) for different \( x \) overlap, then even at the center of \( q_\phi(z|x^{(i)}) \) of one training point \( x^{(i)} \), the optimal decoder will be an average of both \( x^{(i)} \) and other training points \( x^{(j)} \), weighted by \( w_i(z) \) and \( w_j(z) \). Rezende & Viola [2018] have shown that weighted average like this is likely to cause poor reconstruction loss (1 in Eq. (3)) and “blurry reconstruction”. Besides poor reconstructions, when \( x \) is high-dimensional, the reconstruction loss would typically dominate ELBO, suggesting that good reconstruction loss is necessary for obtaining good ELBO.

To enlarge (1), it is a crucial goal to appropriately reduce the overlapping of \( q_\phi(z|x) \) for different \( x \). For Gaussian posterior, this seems to involve shrinking the standard deviations (std) of most \( q_\phi(z|x) \),
since when the mean of every \( q_o(z|x) \) is determined, smaller stds for the majority of \( q_o(z|x) \) would effectively result in less overlapping.

When the prior \( p_B(z) \) is fixed (such as unit Gaussian), however, smaller stds for \( q_o(z|x) \) are likely to induce “holes” on the aggregated posterior, causing \( q_o(z) \) to be more dissimilar with the prior, enlarging (2) in Eq. (5). Also, less overlapping among the Gaussian posteriors \( q_o(z|x) \) would correspond to higher mutual information (3) in Eq. (5) (Hoffman & Johnson, 2016; Mathieu et al., 2019). As a result, with a fixed prior, there should exist a trade-off between (1) and (2) + (3). In fact, such a trade-off has already been discussed in the context of \( \beta \)-VAE (Alemi et al., 2018).

It should be a desirable property that the overlapping among \( q_o(z|x) \) would be large for similar \( x \), and conversely, small for dissimilar \( x \). We shall use the term “appropriate overlapping” to denote such property. We believe that the trade-off between the reconstruction loss (1) and the mutual information (3) should be one important ingredient for reaching the “appropriate overlapping”, as implied by their respective meanings. However, the effect of (2) is not so clear. We suspect that, in many cases, this term may exhibit over-regularization, resulting in too large overlapping, and further unsatisfactory reconstruction loss. In this paper, we choose to use learned RealNVP prior to reduce the effect from (2), such that the trade-off would occur mainly between (1) and (3). We hope this will bring us closer to the “appropriate overlapping”, than using the fixed unit Gaussian prior.

In conclusion, when \( p_B(x|z) \) is Bernoulli, we hypothesize that learning the prior can encourage “appropriate overlapping” among Gaussian posteriors, which is vital for good reconstructions and NLLs. Rezende & Viola (2018) has proved when \( p_B(x|z) = \mathcal{N}(\mu_0(z), \sigma^2) \) with fixed constant \( \sigma \), the optimal decoder is also \( \mu_0(z) = \sum_i w_i(z) x_i \), thus our analysis also holds in such situation.

5 Experiments

5.1 Setup

Datasets We use four datasets in our experiments: statically and dynamically binarized MNIST (Larochelle & Murray, 2011), Salakhutdinov & Murray (2008) (denoted as StaticMNIST and MNIST in our paper, respectively), FashionMNIST (Xiao et al., 2017) and Omniglot (Lake et al., 2015). Details of these datasets can be found in Appendix B.1.

Models We perform systematically controlled experiments, using the following VAE variants: (1) DenseVAE, with dense layers; (2) ConvVAE, with convolutional layers; (3) ResnetVAE, with ResNet layers (Zagoruyko & Komodakis, 2016); and (4) PixelVAE (Gulrajani et al., 2017), with several PixelCNN layers on top of the ResnetVAE decoder. For RealNVP priors and posteriors, we use \( K \) blocks of invertible mappings (\( K \) is called flow depth hereafter), while each block contains an invertible dense, a dense coupling layer, and an actnorm (Dinh et al., 2017; Kingma & Dhariwal, 2018). The dimensionality of \( z \) are 40 for StaticMNIST and MNIST, while 64 for FashionMNIST and Omniglot. More details are in Appendix B.2. We also conduct experiments with ResnetHVAE and PixelHVAE (Tomczak & Welling, 2018), see Appendix B.10.

Training and evaluation Unless specified, all experiments are repeated for 3 times to report metric means. We use Adam (Kingma & Ba, 2015) and adopt warm up (KL annealing) (Bowman et al., 2016) to train all models. We perform early-stopping using negative log-likelihood (NLL) on validation set, to prevent over-fitting on StaticMNIST and on all datasets with PixelVAE. We use 1,000 samples to estimate NLL and other metrics on test set. See Appendix B.3 for more details.

5.2 Quantitative results of VAE with RealNVP Prior and One Latent Variable Compared to Deep Hierarchical VAEs and Other Previous Works

In Tables 1 and 2, we compare ResnetVAE and PixelVAE with RealNVP prior to other approaches on StaticMNIST and MNIST. See Tables B.7 and B.8 for results on Omniglot and FashionMNIST, which have a similar trend. All models except ours and that of Huang et al. (2017) used at least 2 latent variables.

Our ResnetVAE with RealNVP prior, \( K = 50 \) is second only to BIVA among all models without PixelCNN decoder, and ranks the first among all models with PixelCNN decoder. On MNIST, the NLL of our model is very close to BIVA, while the latter used 6 latent variables and very complicated
Table 1: Test NLL on StaticMNIST. “†” indicates a hierarchical model with 2 latent variables, while “‡” indicates at least 3 latent variables.

<table>
<thead>
<tr>
<th>Model</th>
<th>NLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Models without PixelCNN decoder</td>
<td></td>
</tr>
<tr>
<td>ConvHV AE + Lars prior† (Bauer &amp; Mnih, 2019)</td>
<td>81.70</td>
</tr>
<tr>
<td>ConvHV AE + VampPrior† (Tomczak &amp; Welling, 2018)</td>
<td>81.09</td>
</tr>
<tr>
<td>ResConv + RealNVP prior (Huang et al., 2017)</td>
<td>81.44</td>
</tr>
<tr>
<td>VAE + IAF† (Kingma et al., 2016)</td>
<td>79.88</td>
</tr>
<tr>
<td>BIVA‡ (Maaløe et al., 2019)</td>
<td>78.59</td>
</tr>
<tr>
<td>ConvVAE + RNVP $p(z), K = 50$</td>
<td>80.09</td>
</tr>
<tr>
<td>ResnetVAE + RNVP $p(z), K = 50$</td>
<td>79.84</td>
</tr>
</tbody>
</table>

| Models with PixelCNN decoder               |      |
| VLAE‡ (Chen et al., 2017)                  | 79.03 |
| PixelHV AE + VampPrior† (Tomczak & Welling, 2018) | 79.75 |
| PixelVAE + RNVP $p(z), K = 50$              | 78.59 |

Table 2: Test NLL on MNIST. “†” and “‡” has the same meaning as Table 1.

<table>
<thead>
<tr>
<th>Model</th>
<th>NLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Models without PixelCNN decoder</td>
<td></td>
</tr>
<tr>
<td>ConvHV AE + Lars prior† (Bauer &amp; Mnih, 2019)</td>
<td>80.30</td>
</tr>
<tr>
<td>ConvHV AE + VampPrior† (Tomczak &amp; Welling, 2018)</td>
<td>79.75</td>
</tr>
<tr>
<td>VAE + IAF† (Kingma et al., 2016)</td>
<td>79.10</td>
</tr>
<tr>
<td>BIVA‡ (Maaløe et al., 2019)</td>
<td>78.41</td>
</tr>
<tr>
<td>ConvVAE + RNVP $p(z), K = 50$</td>
<td>78.61</td>
</tr>
<tr>
<td>ResnetVAE + RNVP $p(z), K = 50$</td>
<td>78.49</td>
</tr>
</tbody>
</table>

| Models with PixelCNN decoder               |      |
| VLAE‡ (Chen et al., 2017)                  | 78.53 |
| PixelVAE† (Gulrajani et al., 2017)         | 79.02 |
| PixelHV AE + VampPrior† (Tomczak & Welling, 2018) | 78.45 |
| PixelVAE + RNVP $p(z), K = 50$              | 78.12 |

5.3 Qualitative Results

Figure 1 samples images from ResnetVAE with/without RealNVP prior. Compared to standard ResnetVAE, ResnetVAE with RealNVP prior produces fewer digits that are hard to interpret. The last column of each 6x6 grid shows the training set images, most similar to the second-to-last column in pixel-wise L2 distance.

Figure 2: Interpolations of $z$ from ResnetVAE, between the centers of $q_\phi(z|x)$ of two training points, and heatmaps of $\log p_\lambda(z)$. The left- and right-most columns are the training points.
Under review as a conference paper at ICLR 2020

Table 3: Average test NLL (lower is better) of different models, with Gaussian prior & Gaussian posterior (“standard”), Gaussian prior & RealNVP posterior (“RNVP $q(z|x)$”), and RealNVP prior & Gaussian posterior (“RNVP $p(z)$”). Flow depth $K = 20$.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>DenseVAE</th>
<th>ResnetVAE</th>
<th>PixelVAE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$q(z</td>
<td>x)$</td>
<td>$p(z)$</td>
</tr>
<tr>
<td>StaticMNIST</td>
<td>88.84</td>
<td>86.07</td>
<td>84.87</td>
</tr>
<tr>
<td>MNIST</td>
<td>84.48</td>
<td>82.53</td>
<td>80.43</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>228.60</td>
<td>227.79</td>
<td>226.11</td>
</tr>
<tr>
<td>Omniglot</td>
<td>106.42</td>
<td>102.97</td>
<td>102.19</td>
</tr>
</tbody>
</table>

Table 4: Test NLL of ResnetVAE, with RealNVP posterior (“RNVP $q(z|x)$”), RealNVP prior (“RNVP $p(z)$”), and RealNVP prior & posterior (“both”). Flow depth $K = 20$.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>ResnetVAE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$q(z</td>
</tr>
<tr>
<td>StaticMNIST</td>
<td>80.97</td>
</tr>
<tr>
<td>MNIST</td>
<td>79.53</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>225.02</td>
</tr>
<tr>
<td>Omniglot</td>
<td>94.30</td>
</tr>
</tbody>
</table>

Table 5: Average number of active units of ResnetVAE, with standard prior & posterior (“standard”), RealNVP posterior (“RNVP $q(z|x)$”), and RealNVP prior (“RNVP $p(z)$”).

<table>
<thead>
<tr>
<th>Datasets</th>
<th>ResnetVAE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$q(z</td>
</tr>
<tr>
<td>StaticMNIST</td>
<td>30</td>
</tr>
<tr>
<td>MNIST</td>
<td>25.3</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>27</td>
</tr>
<tr>
<td>Omniglot</td>
<td>59.3</td>
</tr>
</tbody>
</table>

in pixel-wise L2 distance. There are differences between the last two columns, indicating our model is not just memorizing the training data. More samples are in Appendix B.8.

5.4 Ablation Study

RealNVP prior leads to substantially lower NLLs than standard VAE and RealNVP posterior. Table 3 shows the NLLs of DenseVAE, ResnetVAE and PixelVAE with flow depth $K = 20$. The NLLs of ConvVAE (with similar trends as ResnetVAE) and the standard deviations of all models are reported in Table B.9 while the NLLs of ResnetVAE with different $K$ (range from 1 to 50) can be found in Table B.11. We can see that RealNVP prior consistently outperforms standard VAE and RealNVP posterior in test NLL, with as large improvement as about 2 nats (compared to standard ResnetVAE) or 1 nat (compared to ResnetVAE with RealNVP posterior) on ResnetVAE, and even larger improvement on DenseVAE. The improvement is not so significant on PixelVAE, likely due to the less information encoded in the latent variable of PixelVAE (Gulrajani et al., 2017).

Using both RealNVP prior and posterior shows no significant advantage over using only RealNVP prior, as shown in Table 4 although the total flow depth with both RealNVP prior and posterior is twice as large as the flow depth with RealNVP prior only.

Active units Table 5 counts the active units (Burda et al., 2016) of different ResnetVAEs, which quantifies the number of latent dimensions used for encoding information from input data. We can see that, both RealNVP prior and posterior can make all units of a ResnetVAE to be active (which is in sharp contrast to standard VAE). However, RealNVP prior can in fact result in substantially better reconstruction loss than RealNVP posterior (see Table B.14). This indicates that, the good regularization effect, “a learned RealNVP prior can lead to more active units than a fixed prior” (Tomczak & Welling, 2018; Bauer & Mnih, 2019), is not the main cause of the huge improvement in NLLs, especially for the improvement of RealNVP prior over RealNVP posterior.

5.5 Reconstruction Loss and Posterior Overlapping

Better reconstruction loss, but larger KL divergence. In Table 6, ELBO and reconstruction loss (“recons”, which is $\mathbb{1}$ in Eq. (5)) of ResnetVAE with RealNVP prior are substantially higher.
than standard ResnetVAE, just as the trend of test log-likelihood (LL) in Table 3. Metrics of other
pairs of $q$ are larger. Larger distances between closest pairs of $q(x|z)$, and (right) normalized distances. See Appendix B.4 for formulation.

Figure 3: Histograms of: (left) per-dimensional stds of $q(x|z)$; (middle) distances between closest pairs of $q(x|z)$, and (right) normalized distances. See Appendix B.4 for formulation.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>standard elbo</th>
<th>standard recons</th>
<th>kl</th>
<th>kl ___</th>
<th>RealNVP $p(z)$ elbo</th>
<th>RealNVP $p(z)$ recons</th>
<th>kl</th>
<th>kl ___</th>
</tr>
</thead>
<tbody>
<tr>
<td>StaticMNIST</td>
<td>-87.61</td>
<td>-60.09</td>
<td>27.52</td>
<td>4.67</td>
<td>-82.85</td>
<td>-54.32</td>
<td>28.54</td>
<td>2.87</td>
</tr>
<tr>
<td>MNIST</td>
<td>-84.62</td>
<td>-58.70</td>
<td>25.92</td>
<td>3.55</td>
<td>-80.34</td>
<td>-53.64</td>
<td>26.70</td>
<td>1.76</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>-228.91</td>
<td>-208.94</td>
<td>19.96</td>
<td>2.74</td>
<td>-225.97</td>
<td>-204.66</td>
<td>21.31</td>
<td>1.88</td>
</tr>
<tr>
<td>Omniglot</td>
<td>-104.87</td>
<td>-66.98</td>
<td>37.89</td>
<td>7.88</td>
<td>-99.60</td>
<td>-61.21</td>
<td>38.39</td>
<td>5.99</td>
</tr>
</tbody>
</table>

Table 6: Average test ELBO (“elbo”), reconstruction loss (“recons”), $E_{\phi(x)} D_{KL}(q_\phi(x|z)||p_\lambda(z))$ (“kl”), and $E_{\phi(x)} D_{KL}(q_\phi(x|z)||p_\theta(z))$ (“kl \_\_\_”) of ResnetVAE with different priors.
on a particular dataset, when traded between reconstruction loss and KL divergence. Given this, we conclude that learned prior can lead to a “better” trade-off from the perspective of RD curve. We also plotted the average normalized distance of β-ResnetVAE trained with different β and prior flow depth K in Fig. 5. The average std of $q_\phi(z|x)$ can be found at Fig. B.4. Learned prior can encourage less posterior overlapping than unit Gaussian prior for various β, not only for β = 1.

6 RELATED WORK

Learned priors, as a natural choice for the conditional priors of intermediate variables, have long been unintentionally used in hierarchical VAEs (Rezende et al., 2014; Sønderby et al., 2016; Kingma et al., 2016; Maaløe et al., 2019). A few works were proposed to enrich the prior, e.g., Gaussian mixture priors (Nalisnick et al., 2016; Dilokthanakul et al., 2016), Bayesian non-parametric priors (Nalisnick & Smyth, 2017; Goyal et al., 2017), and auto-regressive priors (Gulrajani et al., 2017; Chen et al., 2017), without the awareness of its relationship with the aggregated posterior, until the analysis made by Hoffman & Johnson (2016). Since then, attempts have been made in matching the prior to aggregated posterior, by using RealNVP (Huang et al., 2017), variational mixture of posteriors (Tomczak & Welling, 2018), learned accept/reject sampling (Bauer & Mnih, 2019), and kernel density trick (Takahashi et al., 2019). However, none of these works recognized the improved reconstruction loss induced by learned prior. Furthermore, they did not show that learned prior with just one latent variable can achieve comparable results to those of many deep hierarchical VAEs.

The trade-off between reconstruction loss and KL divergence was discussed in the context of β-VAE (Higgins et al., 2017; Alemi et al., 2018; Mathieu et al., 2019), however, they did not further discuss the impact of a learned prior on this trade-off. Mathieu et al. (2019) also discussed the posterior overlapping, but only within the β-VAE framework, thus was only able to control the degree of overlapping globally, without considering the local dissimilarity between x.

7 CONCLUSION

In this paper, using learned RealNVP prior with just one latent variable in VAE, we managed to achieve test NLLs comparable to very deep state-of-the-art hierarchical VAE, outperforming many previous works of complex hierarchical VAEs equipped with rich priors/posteriors. We hypothesized that, when coupled with Gaussian posteriors, the learned prior can encourage appropriate posterior overlapping, which is likely to benefit the reconstruction loss and lower-bound, supported by our experimental results. We showed that with learned RealNVP prior, β-VAE can have better rate-distortion curve (Alemi et al., 2018) than with fixed Gaussian prior. We believe this paper is an important step towards shallow VAEs with learned prior and a small number of latent variables, which potentially can be more scalable to large datasets than those deep hierarchical VAEs.
REFERENCES


A PROOF FOR PROPOSITION 1

We first re-state Proposition 1.

Proposition. Given a finite number of discrete training data, i.e., \( p^*(x) = \frac{1}{N} \sum_{i=1}^{N} \delta(x - x^{(i)}) \), if \( p_\theta(x|z) = \text{Bernoulli}(\mu_\theta(z)) \), where the Bernoulli mean \( \mu_\theta(z) \) is produced by the decoder, and \( 0 < \mu_\theta(z) < 1 \) for each of its \( k \)-th dimensional output, then the optimal decoder \( \mu_\theta(z) \) is:

\[
\mu_\theta(z) = \sum_i w_i(z) x^{(i)}, \quad \text{where} \quad w_i(z) = \frac{q_\theta(z|x^{(i)})}{\sum_j q_\theta(z|x^{(j)})} \quad \text{and} \quad \sum_i w_i(z) = 1
\]

We shall need the Euler’s equation for the calculus of variations (see Gelfand & Silverman (2000, pages 15, 24, and 35)).

Theorem A.1. Let \( \mathcal{L}[f] \) be a functional of the form:

\[
\mathcal{L}[f] = \int_a^b F(x,f,\partial_x f) \, dx
\]

defined on the set of functions \( f(x) \) which satisfies the boundary conditions \( y(a) = A, y(b) = B \), then a necessary condition for \( \mathcal{L}[f] \) to have an extremum for a given function \( f(x) \) is that, \( f(x) \) satisfy Euler’s equation (Gelfand & Silverman 2000, page 15):

\[
\frac{\partial F}{\partial f} - \frac{d}{dx} \frac{\partial F}{\partial (\partial_x f)} = 0 \tag{A.1}
\]

where we use \( \partial_x f \) to denote \( \frac{df}{dx} \).

For multi-variate function \( f(x_1,x_2) \) and the functional:

\[
\mathcal{L}[f] = \iint_R F(x_1,x_2,f,\partial_{x_1} f,\partial_{x_2} f) \, dx_1 \, dx_2
\]

where \( R \) is a closed area, the Euler’s equation becomes (Gelfand & Silverman 2000, page 24):

\[
\frac{\partial F}{\partial f} - \frac{\partial}{\partial x_1} \frac{\partial F}{\partial (\partial_{x_1} f)} - \frac{\partial}{\partial x_2} \frac{\partial F}{\partial (\partial_{x_2} f)} = 0 \tag{A.2}
\]

For vector value function \( f(x) = [f^1(x), \ldots, f^n(x)] \) and the functional:

\[
\mathcal{L}[f] = \int_a^b F(x,f^1,\ldots,f^n,\partial_x f^1,\ldots,\partial_x f^n) \, dx
\]

a necessary condition for \( f(x) \) to be an extremal of \( \mathcal{L}[f] \) is that, every \( f^i(x) \) satisfies Euler’s equation (Gelfand & Silverman 2000, page 35).

With some straightforward extensions, we can also get to the Euler’s equation for multi-variate vector value functions.

The original Euler’s equation requires the integral to be definite, whereas a neural network is defined on the whole \( \mathbb{R}^n \) space. However, in practice, the input values (i.e., the training and test data) are usually within limited range, and the norm of the neural network parameters are not too large. As a result, there should exist a sufficiently large number \( R \), such that all intermediate outputs are within the closed area \([-R, R]^n\). In this sense, the definite and indefinite integral would not have too much difference. We shall always use the indefinite integral notations in the following proof.

Finally, we get to the proof of Proposition 1.

Proof. To apply calculus of variations, we need to substitute the parameterized, bounded \( \mu_\theta(z) \) with a non-parameterized, unbounded mapping. Since \( 0 < \mu_\theta(z) < 1 \), and \( \mu_\theta(z) \) is produced by neural network, which ensures \( \mu_\theta(z) \) is a continuous mapping, then \( \forall \theta \), there exists unbounded \( t(z) \), s.t.

\[
\mu_\theta^k(z) = \frac{\exp(t^k(z))}{1 + \exp(t^k(z))} \quad \text{and} \quad t^k(z) = \log \mu_\theta^k(z) - \log(1 - \mu_\theta^k(z))
\]
and for all continuous mapping $t(z)$, there also exists $\mu_\theta(z)$, satisfying the above equations. In fact, this substitution is also adopted in the actual implementation of our models.

The probability of $p_\theta(x|z)$ is given by:

$$p_\theta(x|z) = \text{Bernoulli}(\mu_\theta(z)) = \prod_k \left( \mu_\theta(z)^{x_k} (1 - \mu_\theta(z))^{1-x_k} \right)$$

Then we have:

$$
\log p_\theta(x|z) = \sum_k \left\{ x_k \log \mu_\theta(z)^k + (1 - x_k) \log (1 - \mu_\theta(z)) \right\} \\
= \sum_k \left\{ x_k t^k(z) - x_k \log \left[ 1 + \exp(t^k(z)) \right] \right\} \\
= \sum_k \left\{ x_k t^k(z) - \log \left[ 1 + \exp(t^k(z)) \right] \right\}
$$

The training objective $\mathcal{L}$ can be then formulated as a functional on $t(z)$:

$$
\mathcal{L}[t] = \mathbb{E}_{p^*(x)} \mathbb{E}_{q_\phi(z|x)} \left[ \log p_\theta(x|z) + \log \frac{p_\theta(z)}{q_\phi(z|x)} \right] \\
= \int \int p^*(x) q_\phi(z|x) \cdot \left( \log p_\theta(x|z) + \log \frac{p_\theta(z)}{q_\phi(z|x)} \right) dz dx \\
= \int F(z, t) dz
$$

where $F(z, t)$ is:

$$
F(z, t) = \int p^*(x) q_\phi(z|x) \left\{ \sum_k \left\{ x_k t^k(z) - \log \left[ 1 + \exp(t^k(z)) \right] \right\} + \log \frac{p_\theta(z)}{q_\phi(z|x)} \right\} dx
$$

According to Euler’s equation (Gelfand & Silverman 2000 page 14 and 35), the necessary condition for $\mathcal{L}[t]$ to have an extremum for a given $t(z)$ is that, $t(z)$ satisfies $\partial F/\partial t^k = 0, \forall k$. Thus we have:

$$
\frac{\partial F}{\partial t^k} = 0 \implies \int p^*(x) q_\phi(z|x) \left[ x_k - \frac{\exp(t^k(z))}{1 + \exp(t^k(z))} \right] dx = 0 \\
\implies \int p^*(x) q_\phi(z|x) \left[ x_k - \mu_\theta(z) \right] dx = 0 \\
\implies \sum_i q_\phi(z|x^{(i)}) \left[ x_k^{(i)} - \mu_\theta(z) \right] = 0 \\
\implies \sum_i q_\phi(z|x^{(i)}) x_k^{(i)} = \left( \sum_i q_\phi(z|x^{(i)}) \right) \mu_\theta(z) \\
\implies \mu_\theta(z) = \frac{\sum_i q_\phi(z|x^{(i)}) x_k^{(i)} }{\sum_i q_\phi(z|x^{(i)})} = \sum_i w_i(z) x^{(i)}.
$$

Rezende & Viola (2018) have proved that when $p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \sigma^2 I)$, where $\sigma$ is a global fixed constant, the optimal decoder $\mu_\theta(z) = \sum_i w_i(z) x^{(i)}$, which is exactly the same as our conclusion. Kosca et al. (2018) have proved that the gradient of Bernoulli$(\mu_\theta(z))$ is the same as that of $\mathcal{N}(\mu_\theta(z), \sigma^2 I)$ when $\sigma = 1$, but they did not calculate out the optimal decoder. We push forward both these works.
B EXPERIMENTAL DETAILS

B.1 DATASETS

MNIST  MNIST is a 28x28 grayscale image dataset of hand-written digits, with 60,000 data points for training and 10,000 for testing. When validation is required for early-stopping, we randomly split the training data into 50,000 for training and 10,000 for validation. Since we use Bernoulli $p_θ(x|z)$ to model these images in VAE, we binarize these images by the method in (Salakhutdinov & Murray, 2008): each pixel value is randomly set to 1 in proportion to its pixel intensity. The training and validation images are re-binarized at each epoch. However, the test images are binarized beforehand for all experiments. We binarize each test image 10 times, and use all these 10 binarized data points in evaluation. This method results in a 10 times larger test set, but we believe this can help us to obtain a more objective evaluation result.

StaticMNIST  StaticMNIST (Larochelle & Murray, 2011) is a pre-binarized MNIST image dataset, with the original 60,000 training data already split into 50,000 for training and 10,000 for validation. We always use validation set for early-stopping on StaticMNIST. Meanwhile, since StaticMNIST has already been binarized, the test set is used as-is without 10x enlargement.

FashionMNIST  FashionMNIST (Xiao et al., 2017) is a recently proposed image dataset of grayscale fashion products, with the same specification as MNIST. We thus use the same training-validation split and the same binarization method just as MNIST.

Omniglot  Omniglot (Lake et al., 2015) is a 28x28 grayscale image dataset of hand-written characters. We use the preprocessed data from (Burda et al., 2016), with 24,345 data points for training and 8,070 for testing. When validation is required, we randomly split the training data into 20,345 for training and 4,000 for validation. We use dynamic binarization on Omniglot just as MNIST.

B.2 NETWORK ARCHITECTURES

Notations  In order to describe the detailed architecture of our models, we will introduce auxiliary functions to denote network components. A function $h_φ(x)$ should denote a sub-network in $q_φ(z|x)$, with a subset of $φ$ as its own learnable parameters. For example, if we write $q_φ(z|x) = q_φ(z|h_φ(x)) = \mathcal{N}(\mathbf{μ}_φ(h_φ(x)), \mathbf{σ}_φ^2(h_φ(x))\mathbf{I})$, it means that the posterior $q_φ(z|x)$ is a Gaussian, whose mean and standard deviation are derived by one shared sub-network $h_φ(x)$ and two separate sub-networks $μ_φ(·)$ and $σ_φ(·)$, respectively.

The structure of a network is described by composition of elementary neural network layers.

Linear[$k$] indicates a linear dense layer with $k$ outputs. Dense[$k$] indicates a non-linear dense layer. $a \to b$ indicates a composition of $a$ and $b$, e.g., Dense[$m$] $\to$ Dense[$n$] indicates two successive dense layers, where the first layer has $m$ outputs and the second layer has $n$ outputs. These two dense layers can also be abbreviated as Dense[$m \to n$].

Conv[$H \times W \times C$] denotes a non-linear convolution layer, whose output shape is $H \times W \times C$, where $H$ is the height, $W$ is the width and $C$ is the channel size. As an abbreviation, Conv[$H_1 \times W_1 \times C_1 \to H_2 \times W_2 \times C_2$] denotes two successive non-linear convolution layers. Resnet[·] denotes non-linear resnet layer(s) (Zagoruyko & Komodakis, 2016). All Conv and Resnet layers by default use 3x3 kernels, unless the kernel size is specified as subscript (e.g., Conv$_{1 \times 1}$ denotes a 1x1 convolution layer). The strides of Conv and Resnet layers are automatically determined by the input and output shapes, which is 2 in most cases. LinearConv[·] denotes linear convolution layer(s).

DeConv[·] and DeResnet[·] denote deconvolution and deconvolutional resnet layers, respectively.

PixelCNN[·] is a PixelCNN layer proposed by Salimans et al. (2017). It uses resnet layers, instead of convolution layers. Details can be found in its original paper.

Flatten indicates to reshape the input 3-d tensor into a vector, while UnFlatten[$H \times W \times C$] indicates to reshape the input vector into a 3-d tensor of shape $H \times W \times C$. Concat[$a, b$] indicates to concat the output of $a$ and $b$ along the last axis.
The Gaussian posterior thus requiring higher dimensional latent variables. Is because we think the latter two datasets are conceptually more complicated than the former two, is chosen to be 40 on MNIST and StaticMNIST, while 64 on FashionMNIST and Omniglot. This $z$ or not the model is convolutional. The latent variable $x$ is a vector, whose number of dimensions is chosen to be 40 on MNIST and StaticMNIST, while 64 on FashionMNIST and Omniglot. This is because we think the latter two datasets are conceptually more complicated than the former two, thus requiring higher dimensional latent variables.

The observed variable $x$ (i.e., the input image) is always a 3-d tensor, with shape $H \times W \times C$, whether or not the model is convolutional. The latent variable $z$ is a vector, derived as:

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(h_\phi(x)), \sigma^2_\phi(h_\phi(x))) I$$

$$\mu_\phi(h_\phi(x)) = h_\phi(x) \rightarrow \text{Linear}[\text{Dim}(z)]$$

$$\log \sigma_\phi(h_\phi(x)) = h_\phi(x) \rightarrow \text{Linear}[\text{Dim}(z)]$$

Note we make the network to produce $\log \sigma_\phi(h_\phi(x))$ instead of directly producing $\sigma_\phi(h_\phi(x))$. $h_\phi(x)$ is the hidden layers, varying among different models.

For binarized images, we use Bernoulli conditional distribution $p_\theta(x|z)$, derived as:

$$p_\theta(x|z) = \text{Bernoulli}[\mu_\phi(h_\theta(z))]$$

For DenseVAE:

$$\log \frac{\mu_\theta(h_\theta(z))}{1 - \mu_\theta(h_\theta(z))} = h_\theta(z) \rightarrow \text{Linear}[784] \rightarrow \text{UnFlatten}[28 \times 28 \times 1]$$

And for others (i.e., ConvVAE, ResnetVAE and PixelVAE):

$$\log \frac{\mu_\theta(h_\theta(z))}{1 - \mu_\theta(h_\theta(z))} = h_\theta(z) \rightarrow \text{LinearConv}_{1 \times 1}[28 \times 28 \times 1]$$

Note we make the network to produce $\log \frac{\mu_\theta(h_\theta(z))}{1 - \mu_\theta(h_\theta(z))}$, the logits of Bernoulli distribution, instead of producing the Bernoulli mean $\mu_\theta(h_\theta(z))$ directly.

**DenseVAE** $h_\phi(x)$ and $h_\theta(z)$ of DenseVAE are composed of dense layers, formulated as:

$$h_\phi(x) = x \rightarrow \text{Flatten} \rightarrow \text{Dense}[500 \rightarrow 500]$$

$$h_\theta(z) = z \rightarrow \text{Dense}[500 \rightarrow 500]$$

**Conv/ResnetVAE** $h_\phi(x)$ and $h_\theta(z)$ of ConvVAE are composed of (de)convolutional layers, while those of ResnetVAE consist of (deconvolutional) resnet layers. We only describe the architecture of ResnetVAE here. The structure of ConvVAE can be easily obtained by replacing all (deconvolutional) resnet layers with (de)convolution layers:

$$h_\phi(x) = x \rightarrow \text{Resnet}[28 \times 28 \times 32 \rightarrow 28 \times 28 \times 32 \rightarrow 14 \times 14 \times 64 \rightarrow 14 \times 14 \times 64 \rightarrow 7 \times 7 \times 64 \rightarrow 7 \times 7 \times 16] \rightarrow \text{Flatten}$$

$$h_\theta(z) = z \rightarrow \text{Dense}[784] \rightarrow \text{UnFlatten}[7 \times 7 \times 16] \rightarrow \text{DeResnet}[7 \times 7 \times 64 \rightarrow 14 \times 14 \times 64 \rightarrow 14 \times 14 \times 64 \rightarrow 28 \times 28 \times 32 \rightarrow 28 \times 28 \times 32]$$

**General configurations** All non-linear layers use leaky relu activation.
PixelVAE $h_\phi(x)$ of PixelVAE is exactly the same as ResnetVAE, while $h_\theta(z)$ is derived as:

$$
\begin{align*}
    h_\phi(z) &= \text{Concat}[x, \tilde{h}_\phi(z)] \\
    &\rightarrow \text{PixelCNN}[28 \times 28 \times 33 \rightarrow 28 \times 28 \times 33] \\
    &\rightarrow \text{28} \times \text{28} \times \text{33}
\end{align*}
$$

$$
\begin{align*}
    \tilde{h}_\phi(z) &= z \rightarrow \text{Dense[784]} \rightarrow \text{UnFlatten}[7 \times 7 \times 16] \\
    &\rightarrow \text{DeResnet}[7 \times 7 \times 64 \rightarrow 14 \times 14 \times 64] \\
    &\rightarrow \text{14} \times \text{14} \times \text{64} \rightarrow \text{28} \times \text{28} \times \text{32} \\
    &\rightarrow \text{28} \times \text{28} \times \text{32}
\end{align*}
$$

As Salimans et al. [2017], we use dropout in PixelCNN layers, with rate 0.5.

RealNVP The RealNVP consists of $K$ blocks, while each block consists of an invertible dense, a coupling layer, and an actnorm. The RealNVP mapping for prior, i.e. $f_\lambda(z)$, can be formulated as:

$$
\begin{align*}
    f_{k}(h_k) &= h_k \rightarrow \text{InvertibleDense} \rightarrow \text{CouplingLayer} \rightarrow \text{ActNorm} \\
    \text{CouplingLayer}(u) &= \text{Concat}[u_t, u_r \odot \text{Sigmoid}(s_\lambda(h_{\lambda,k}(u_t))) + t_\lambda(h_{\lambda,k}(u_t))] \\
    h_{\lambda,k}(u_t) &= u_t \rightarrow \text{Dense[256]} \\
    s_\lambda(h_{\lambda,k}(u_t)) &= h_{\lambda,k}(u_t) \rightarrow \text{Linear[Dim}(u)] \\
    t_\lambda(h_{\lambda,k}(u_t)) &= h_{\lambda,k}(u_t) \rightarrow \text{Linear[Dim}(u_r)]
\end{align*}
$$

where $u_t = u_{0:[\text{Dim}(u)/2]}$ is the left half of $u$, and $u_r = u_{[\text{Dim}(u)/2]:	ext{Dim}(u)}$ is the right half.

The RealNVP posterior, derived from the original Gaussian posterior $q_\phi(w|x)$, is denoted as $q_{\phi,\eta}(z|x)$, and formulated as:

$$
\begin{align*}
    q_{\phi,\eta}(z|x) &= q_\phi(w|x) \left| \frac{\partial f_\eta(w)}{\partial w} \right|^{-1} \\
    z &= f_\eta(w)
\end{align*}
$$

where the structure of the RealNVP mapping $f_\eta(w)$ for posterior is exactly the same as $f_\lambda(z)$ for prior. The ELBO for VAE with RealNVP posterior is then simply:

$$
\begin{align*}
    \mathcal{L}(\lambda, \theta, \phi, \eta) &= \mathbb{E}_{p^*(x)} \mathbb{E}_{q_\phi(w|x)} \left[ \log p_\theta(x|f_\eta(w)) + \log p_\lambda(f_\eta(w)) \\
    &- \log q_\phi(w|x) + \log \left| \det \left( \frac{\partial f_\eta(w)}{\partial w} \right) \right| \right] \tag{B.2}
\end{align*}
$$

B.3 Additional Details of Training and Evaluation

General methodology All the mathematical expressions of expectations w.r.t. some distributions are computed by Monte Carlo integration. For example, $\mathbb{E}_{q_\phi(z|x)} [f(z, x)]$ is estimated by:

$$
\mathbb{E}_{q_\phi(z|x)} [f(z, x)] \approx \frac{1}{L} \sum_{i=1}^{L} f(z^{(i)}, x)
$$

where $z^{(i)}$ is one sample from $q_\phi(z|x)$.

Training We use Adam [Kingma & Ba, 2015] to train our models for 2,400 epochs. The batch size is 128 for DenseVAE, ConvVAE and ResnetVAE, and 64 for PixelVAE. On MNIST, FashionMNIST and Omniglot, we set the learning rate to be $10^{-3}$ in the first 800 epochs, $10^{-4}$ in the next 800 epochs, and $10^{-5}$ in the last 800 epochs. On StaticMNIST, we set the learning rate to be $10^{-4}$ in the first 1,600 epochs, and $10^{-5}$ in the last 800 epochs.

L2 regularization with factor $10^{-4}$ is applied on weights of all non-linear hidden layers, i.e., kernels of non-linear dense layers and convolutional layers, in $h_\phi(x)$, $h_\theta(z)$, $f_\lambda(z)$ and $f_\eta(w)$.
The ELBO is estimated by 1 \( z \) sample for each \( x \) in training. We adopt warm-up (KL annealing) (Bowman et al. [2016]). The ELBO using warm-up is formulated as:

\[
\mathcal{L}(x; \lambda, \theta, \phi) = \mathbb{E}_{q_\phi(x|x)} \left[ \log p_\theta(x|z) + \beta \left( \log p_\lambda(z) - \log q_\phi(z|x) \right) \right]
\]

\( \beta \) is increased from 0.01 to 1 linearly in the first 100 epochs, and it remains 1 afterwards. The warm-up ELBO for VAEs with RealNVP priors and posteriors can be obtained by replacing \( p_\lambda(z) \) and \( q_\phi(z|x) \) of the above equation by Eq. (4) and Eq. (B.1), respectively.

We adopt early-stopping using NLL on validation set, to prevent over-fitting on StaticMNIST and Omniglot. For StaticMNIST, we find it overfits after only a few iterations, thus we choose \( M = 16 \), and the number of epochs chosen to be 3,200 epochs on all datasets (starting from Iteration 1b), after the pre-training step (Algorithm B.1).

### Training strategies for \( p_\lambda(z) \)
We consider three training strategies for optimizing Eq. (7):

- **Post-hoc training** [Bauer & Mnih [2019]]: \( q_\phi(z|x) \) and \( p_\theta(x|z) \) are firstly trained w.r.t. the unit Gaussian prior, then \( q_\phi(z|x) \) and \( p_\theta(x|z) \) are fixed and \( p_\lambda(z) \) is in turn optimized. This is the most intuitive training method according to Eq. (1), however, it does not work as well as joint training in terms of test negative log-likelihood (NLL), which is observed both in our experiments (Table B.3) and by Bauer & Mnih (2019).

- **Joint training** [Tomczak & Welling [2018]; Bauer & Mnih [2019]]: \( p_\lambda(z) \) is jointly trained along with \( q_\phi(z|x) \) and \( p_\theta(x|z) \), by directly maximizing Eq. (7).

- **Iterative training**: Proposed by us, we alternate between training \( p_\theta(x|z) & q_\phi(z|x) \) and training \( p_\lambda(z) \), for multiple iterations. The first iteration to train \( p_\theta(x|z) & q_\phi(z|x) \) should use the unit Gaussian prior. Early-stopping should be performed during the whole process if necessary. See Algorithm B.1 for detailed procedure of this strategy. We adopt this method mainly for investigating why post-hoc training does not work as well as joint training.

To train \( p_\lambda(z) \) with post-hoc strategy, we start from a trained VAE, adding RealNVP prior onto it, and optimizing the RealNVP \( f_\lambda(z) \) for 3,200 epochs, with learning rate set to 10\(^{-4}\) in the first 1,600 epochs, and 10\(^{-5}\) in the final 1,600 epochs.

### Algorithm B.1 Pseudocode for iterative training.

```
Algorithm B.1 Pseudocode for iterative training.

Iteration 1a: Train \( q_\phi(z|x) \) and \( p_\theta(x|z) \), with \( p_\lambda(z) = N(0, 1) \).

Iteration 1b: Train \( p_\lambda(z) \) for 2M epochs, with fixed \( q_\phi(z|x) \) and \( p_\theta(x|z) \).

for i = 2 \ldots I do

  Iteration 1a: Train \( q_\phi(z|x) \) and \( p_\theta(x|z) \) for M epochs, with fixed \( p_\lambda(z) \).

  Iteration 1b: Train \( p_\lambda(z) \) for M epochs, with fixed \( q_\phi(z|x) \) and \( p_\theta(x|z) \).

end for
```

Algorithm B.1 is the pseudocode of iterative training strategy. For Iteration 1a, all hyper-parameters are the same with training a standard VAE, where in particular, the number of training epochs is set to 2,400. For Iteration 1b, the learning rate is 10\(^{-3}\) for the first \( M \) epochs, and is 10\(^{-4}\) for the next \( M \) epochs. For all the next iterations, learning rate is always 10\(^{-4}\). The number of iterations \( I \) is chosen to be 16, and the number of epochs \( M \) is chosen to be 100, for MNIST, FashionMNIST and Omniglot. For StaticMNIST, we find it overfits after only a few iterations, thus we choose \( I \) to be 4, and \( M \) to be 400. With these hyper-parameters, \( q_\phi(z|x) \), \( p_\theta(x|z) \) and \( p_\lambda(z) \) are iteratively trained for totally 3,200 epochs on all datasets (starting from Iteration 1b), after the pre-training step (Iteration 1a).

### Regularization term for \( q_\phi(z) \)
In order to compare some metrics (e.g., the active units) of VAE + RealNVP prior with those of standard VAE, we introduce an additional regularization term for \( q_\phi(z) \), such that \( q_\phi(z) \) of VAE using RealNVP prior would have roughly zero mean and unit variance, just as a standard VAE with unit Gaussian prior. The regularization term for \( q_\phi(z) \) (denoted as \( \text{Reg} \ [q_\phi(z)] \)) and the final training objective augmented with the regularization term (denoted as \( \tilde{\mathcal{L}}(x; \lambda, \theta, \phi) \)) are:

\[
\text{Reg} \ [q_\phi(z)] = \frac{1}{\text{Dim}(z)} \sum_{k=1}^{\text{Dim}(z)} \left[ (\text{Mean}(z_k))^2 + (\text{Var}(z_k) - 1)^2 \right]
\]

\[
\tilde{\mathcal{L}}(\lambda, \theta, \phi) = \mathbb{E}_{p(\cdot|x)} \mathcal{L}(x; \lambda, \theta, \phi) + \text{Reg} \ [q_\phi(z)]
\]
where $z_k$ is the $k$-th dimension of $z$, $\text{Dim}(z)$ is the number of dimensions, $\text{Mean}(z_k) = \mathbb{E}_{p^*(x)} \mathbb{E}_{q_\phi(z|x)} [z_k]$ and $\text{Var}(z_k) = \mathbb{E}_{p^*(x)} \mathbb{E}_{q_\phi(z|x)} \left( (z_k - \text{Mean}(z_k))^2 \right)$ are the mean and variance of each dimension.

Table B.1: Avg. Mean($z_k$) and Var($z_k$) of regularized/un-regularized ResNetVAE with RealNVP prior.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>regularized</th>
<th>un-regularized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg. Mean($z_k$)</td>
<td>Avg. Var($z_k$)</td>
</tr>
<tr>
<td>StaticMNIST</td>
<td>-0.02 ± 0.03</td>
<td>0.93 ± 0.01</td>
</tr>
<tr>
<td>MNIST</td>
<td>0.00 ± 0.00</td>
<td>0.98 ± 0.01</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>0.00 ± 0.00</td>
<td>0.98 ± 0.00</td>
</tr>
<tr>
<td>Omniglot</td>
<td>0.00 ± 0.02</td>
<td>0.95 ± 0.00</td>
</tr>
</tbody>
</table>

Table B.1 shows the average $\text{Mean}(z_k)$ and $\text{Var}(z_k)$ of ResNetVAE with RealNVP prior, computed on test data. Average $\text{Mean}(z_k)$ is defined as $\frac{1}{\text{Dim}(z)} \sum_{k=1}^{\text{Dim}(z)} \text{Mean}(z_k)$, while average $\text{Var}(z_k)$ is defined as $\frac{1}{\text{Dim}(z)} \sum_{k=1}^{\text{Dim}(z)} \text{Var}(z_k)$. The means and standard deviations of the above table are computed w.r.t. repeated experiments. Using the regularization term for $q_\phi(z)$ makes the $\text{Mean}(z_k)$ and $\text{Var}(z_k)$ of $z$ samples close to $\mathcal{N}(0, I)$, which is in sharp contrast with the un-regularized case. For fair comparison in Table5 and Fig 3, it is crucial to have $\text{Mean}(z_k)$ and $\text{Var}(z_k)$ close to $\mathcal{N}(0, I)$.

Test NLLs are not reported here, because we find no significant difference between regularized and un-regularized models in terms of test NLLs.

**Evaluation** The negative log-likelihood (NLL), the reconstruction loss and the KL divergence $\mathbb{E}_{p^*(x)} D_{\text{KL}}(q_\phi(z) || p_\lambda(z))$ are estimated with 1,000 $z$ samples from $q_\phi(z|x^{(i)})$ for each $x^{(i)}$ from test data:

$$\text{NLL} \approx \frac{1}{N} \sum_{i=1}^{N} \log \text{MeanExp}_{j=1}^{1000} \left[ \log p_\theta(x^{(i)}|z^{(i,j)}) + \log p_\lambda(z^{(i,j)}) - \log q_\phi(z^{(i,j)})|x^{(i)} \right]$$

$$\text{Reconstruction Loss} \approx \frac{1}{1000N} \sum_{i=1}^{N} \sum_{j=1}^{1000} \left[ \log p_\theta(x^{(i)}|z^{(i,j)}) \right]$$

$$\mathbb{E}_{p^*(x)} D_{\text{KL}}(q_\phi(z) || p_\lambda(z)) \approx \frac{1}{1000N} \sum_{i=1}^{N} \sum_{j=1}^{1000} \left[ \log q_\phi(z^{(i,j)}|x^{(i)}) - \log p_\lambda(z^{(i,j)}) \right]$$

where each $z^{(i,j)}$ is one sample from $q_\phi(z|x^{(i)})$.

$\text{LogMeanExp}_{j=1}^{L} \left[ f(x^{(i)}, z^{(i,j)}) \right]$ is defined as:

$$\text{LogMeanExp}_{j=1}^{L} \left[ f(x^{(i)}, z^{(i,j)}) \right] = f_{\max} + \log \frac{1}{L} \sum_{j=1}^{L} \left[ \exp \left( f(x^{(i)}, z^{(i,j)}) - f_{\max} \right) \right]$$

where $f_{\max} = \max_j f(x^{(i)}, z^{(i,j)})$.

We use 1,000 samples, because the two major previous works of us (i.e., Tomczak & Welling [2018], Bauer & Mnih [2019]) both used this number of samples. It is true that some previous works may have used substantially larger number of samples (e.g., Chen et al. [2017] used 4,096 samples), and according to Grosse et al. [2015], the Monte Carlo estimator for NLL is a stochastic lower-bound, such that larger number of samples typically would lead to lower (yet better) estimated NLLs. However, in order to have fair comparison with Tomczak & Welling [2018, Bauer & Mnih 2019], we decide to stick to 1,000 samples.

**Active units** (Burda et al. [2016]) is defined as the number of latent dimensions whose variance is larger than 0.01. The variance of the $k$-th dimension is formulated as:

$$\text{Var}_k = \text{Var}_{p^*(x)}(\mathbb{E}_{q_\phi(z|x)} [z_k])$$
where $z_k$ is the $k$-th dimension of $z$. We compute $\text{Var}_k$ on the training data, while the inner expectation $\mathbb{E}_{q_\theta(z|x)}[z_k]$ is estimated by drawing 1,000 samples of $z$ for each $x$.

**B.4 Formulation of closest pairs of $q_\phi(z|x)$ and others**

**Closest pairs of $q_\phi(x|i)$** For each $x^{(i)}$ from training data, we find the training point $x^{(j)}$, whose posterior $q_\phi(z|x^{(j)})$ is the closest neighbor to $q_\phi(z|x^{(i)})$:

$$j = \arg\min_{j \neq i} \| \mu_\phi(x^{(j)}) - \mu_\phi(x^{(i)}) \|$$

where $\mu_\phi(x)$ is the mean of $q_\phi(z|x)$, and $\| \cdot \|$ is the L2 norm. These two posteriors $q_\phi(z|x^{(i)})$ and $q_\phi(z|x^{(j)})$ are called a closest pair of $q_\phi(z|x)$.

**Distance of a pair of $q_\phi(z|x)$** The distance $d_{ij}$ of a closest pair $q_\phi(z|x^{(i)})$ and $q_\phi(z|x^{(j)})$ is:

$$d_{ij} = \| \mu_\phi(x^{(i)}) - \mu_\phi(x^{(j)}) \|$$

**Normalized distance of a pair of $q_\phi(z|x)$** For each closest pair $q_\phi(z|x^{(i)})$ and $q_\phi(z|x^{(j)})$, we compute its normalized distance $\tilde{d}_{ij}$ by:

$$\tilde{d}_{ij} = \frac{2d_{ij}}{\text{Std}[i; j] + \text{Std}[j; i]}$$

**B.5 Discussions about $D_{\text{KL}}(q_\phi(z)||p_\lambda(z))$**

The KL divergence between the aggregated posterior and the prior, i.e., $D_{\text{KL}}(q_\phi(z)||p_\lambda(z))$, is first analyzed by [Hoffman & Johnson (2016)](Hoffman & Johnson (2016)) as one component of the ELBO decomposition [B.1]. Since $q_\phi(z) = \int q_\phi(z|x)p^*(x)\,dx$ and $p_\lambda(z) = \int p_\lambda(z|x)p^*(x)\,dx$, [Rosca et al. (2018)](Rosca et al. (2018)) used this KL divergence as a metric to quantify the approximation quality of both $p_\lambda(z|x)$ to $p^*(x)$, and $q_\phi(z|x)$ to $p_\lambda(z|x)$. As we are focusing on learning the prior, $D_{\text{KL}}(q_\phi(z)||p_\lambda(z))$ can in turn be a metric for quantifying whether $p_\lambda(z)$ is close enough to the aggregated posterior $q_\phi(z)$.

The evaluation of $D_{\text{KL}}(q_\phi(z)||p_\lambda(z))$, however, is not an easy task. One way to estimate the KL divergence of two arbitrary distributions is the density ratio trick [Sugiyama et al., 2012, Mescheder et al., 2017], where $D_{\text{KL}}(q_\phi(z)||p_\lambda(z))$ is estimated by a separately trained neural network classifier. However, [Rosca et al. (2018)](Rosca et al. (2018)) revealed that such approach can under-estimate $D_{\text{KL}}(q_\phi(z)||p_\lambda(z))$, and the training may even diverge when $z$ has hundreds of or more dimensions.

Another approach is to directly estimate $D_{\text{KL}}(q_\phi(z)||p_\lambda(z))$ by Monte Carlo integration. The KL divergence can be rewritten into the following form:

$$D_{\text{KL}}(q_\phi(z)||p_\lambda(z)) = \int q_\phi(z) \log \frac{q_\phi(z)}{p_\lambda(z)} \, dz$$

$$= \int \left( \int q_\phi(z|x) p^*(x) \, dx \right) \log \frac{\int q_\phi(z|x') p^*(x') \, dx'}{p_\lambda(z)} \, dz$$

$$= \int p^*(x) \int q_\phi(z|x) \log \frac{\int q_\phi(z|x') p^*(x') \, dx'}{p_\lambda(z)} \, dz \, dx$$

$$= \mathbb{E}_{p^*(x)} \mathbb{E}_{q_\phi(x|x)} \left[ \log \mathbb{E}_{p^*(x')} [q_\phi(z|x')] - \log p_\lambda(z) \right]$$

(B.5)
Rosca et al. (2018) has already proposed a Monte Carlo based algorithm to estimate $D_{KL}(q_\phi(z)||p_\lambda(z))$, in case that the training data is statically binarized. Since we mainly use dynamically binarized datasets, we slightly modified the algorithm according to Eq. (B.5), to allow sampling multiple $x$ from each image. Our algorithm is:

**Algorithm B.2** Pseudocode for estimating $D_{KL}(q_\phi(z)||p_\lambda(z))$ (denoted as marginal_kl) on dynamically binarized dataset, where $\mu$ is the pixel intensities of each original image.

$x_{\text{samples}} = []$

for $\mu$ in training dataset do
  for $i = 1 \ldots n_x$ do
    sample $x$ from Bernoulli($\mu$)
    append $x$ to $x_{\text{samples}}$
  end for
end for

marginal_kl = 0

for $x$ in $x_{\text{samples}}$ do
  for $i = 1 \ldots n_z$ do
    sample $z$ from $q_\phi(z|x)$
    posterior_list = []
    for $x'$ in $x_{\text{samples}}$ do
      append $\log q_\phi(z|x')$ to posterior_list
    end for
    $\log q_\phi(z) = \log \text{LogMeanExp}(\text{posterior_list})$
    marginal_kl = marginal_kl + $\log q_\phi(z) - \log p_\lambda(z)$
  end for
end for

marginal_kl = marginal_kl/($\text{len}(x_{\text{samples}}) \times n_z$)

Surprisingly, we find that increasing $n_x$ will cause the estimated $D_{KL}(q_\phi(z)||p_\lambda(z))$ to decrease on MNIST, see Table B.2. Given that our $n_x = 10$ for all $n_z$, the number of our sampled $z$ (even when $n_x = 1$) is $10 \times 60,000 = 6 \times 10^5$, which should not be too small, since Rosca et al. (2018) only used $10^6 z$ in their experiments. When $n_x = 8$, the number of $z$ is $8 \times 10 \times 60,000 = 4.8 \times 10^8$, which is 4.8x larger than Rosca et al. (2018), not to mention the inner expectation $\mathbb{E}_{p_\lambda(x')}[q_\phi(z|x')]$ is estimated with 8x larger number of $x'$. There should be in total 38.4x larger number of $\log q_\phi(z|x')$ computed for estimating $D_{KL}(q_\phi(z)||p_\lambda(z))$ than Rosca et al. (2018), which has already costed about 3 days on 4 GTX 1080 Ti graphical cards. We believe such a large number of Monte Carlo samples should be sufficient for any algorithm with good behavior.

<table>
<thead>
<tr>
<th>$n_x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{KL}(q_\phi(z)</td>
<td></td>
<td>p_\lambda(z))$</td>
<td>15.279</td>
<td>14.623</td>
<td>14.254</td>
</tr>
</tbody>
</table>

According to the above observation, we suspect there must be some flaw in Algorithm B.2. Because of this, we do not adopt Algorithm B.2 to estimate $D_{KL}(q_\phi(z)||p_\lambda(z))$.

Since no mature method has been published to estimate $D_{KL}(q_\phi(z)||p_\lambda(z))$, we decide not to use $D_{KL}(q_\phi(z)||p_\lambda(z))$ to measure how our learned $p_\lambda(z)$ approximates the aggregated posterior $q_\phi(z)$.

**B.6 Iterative Training Can Lead to Improved Reconstruction Loss and Increased Active Units**

Table B.3 shows the NLLs of *iterative training* and *post-hoc training* with ResnetVAE (see also Appendix B.3). Although still not comparable to *joint training*, both methods can bring large im-
Table B.3: Test NLL of ResnetVAE, with prior trained by: joint training, iterative training, post-hoc training, and standard VAE (“none”) as reference. Flow depth $K = 20$.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>joint</th>
<th>iterative</th>
<th>post-hoc</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>StaticMNIST</td>
<td>79.99</td>
<td>80.63</td>
<td>80.86</td>
<td>82.95</td>
</tr>
<tr>
<td>MNIST</td>
<td>78.58</td>
<td>79.61</td>
<td>79.90</td>
<td>81.07</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>224.09</td>
<td>224.88</td>
<td>225.22</td>
<td>226.17</td>
</tr>
<tr>
<td>Omniglot</td>
<td>93.61</td>
<td>94.43</td>
<td>94.87</td>
<td>96.99</td>
</tr>
</tbody>
</table>

Table B.4: Average reconstruction loss (“recons”), $\mathbb{E}_{p^*(x)} D_{KL}(q_\phi(z|x) \parallel p_\lambda(z))$ (“kl”) and active units (“au”) of ResnetVAE with iteratively trained and post-hoc trained RealNVP priors.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>iterative</th>
<th>post-hoc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>recons</td>
<td>kl</td>
</tr>
<tr>
<td>StaticMNIST</td>
<td>-58.0</td>
<td>26.4</td>
</tr>
<tr>
<td>DynamicMNIST</td>
<td>-57.2</td>
<td>25.1</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>-207.8</td>
<td>19.4</td>
</tr>
<tr>
<td>Omniglot</td>
<td>-63.3</td>
<td>37.9</td>
</tr>
</tbody>
</table>

Table B.5: Test NLL on StaticMNIST. “†” and “‡” has the same meaning as Table 1.

<table>
<thead>
<tr>
<th>Model</th>
<th>NLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Models without PixelCNN decoder</td>
<td></td>
</tr>
<tr>
<td>ConvHVAE + Lars prior†</td>
<td>81.70</td>
</tr>
<tr>
<td>ConvHVAE + VampPrior†</td>
<td>81.09</td>
</tr>
<tr>
<td>ResConv + RealNVP prior</td>
<td>81.44</td>
</tr>
<tr>
<td>VAE + IAF‡ (Kingma et al., 2016)</td>
<td>79.88</td>
</tr>
<tr>
<td>BIVA‡ (Maaløe et al., 2019)</td>
<td>78.59</td>
</tr>
<tr>
<td>ConvVAE + RNVP $p(z)$, $K = 50$</td>
<td>80.09 ± 0.01</td>
</tr>
<tr>
<td>ResnetVAE + RNVP $p(z)$, $K = 50$</td>
<td>79.84 ± 0.04</td>
</tr>
</tbody>
</table>

|Models with PixelCNN decoder       |      |
|VLAE‡ (Chen et al., 2017)          | 79.03|
|PixelHVAE + VampPrior‡             | 79.78|
|PixelVAE + RNVP $p(z)$, $K = 50$   | 79.01 ± 0.03|

Improvement in NLLs over standard VAE. Also, iterative training even further outperforms post-hoc training by a large margin.

Table B.4 shows that, compared with post-hoc training, iterative training can lead to larger reconstruction loss and increased number of active units. Section 5.5 suggests that the learned prior can encourage better trade-off between reconstruction loss and the KL divergence, thus a well-trained prior is essential for obtaining good $q_\phi(z|x)$ and $p_\lambda(x|z)$. However, the prior is in turn determined by $q_\phi(z|x)$ and $p_\lambda(x|z)$, according to Eq. (5). Thus, it is important to alternate between training $p_\lambda(x|z)$ and $q_\phi(z|x)$, until they catch up with each other and reach an equilibrium.

Considering these reasons, it is not surprising why iterative training can result in larger (yet better) reconstruction loss than post-hoc training. It is also clear why joint training works the best: because $p_\lambda(x|z)$ and $q_\phi(z|x)$, and $p_\lambda(z)$, are trained to catch up with each other at every step, thus the training is more sufficient when using joint training than using the other two techniques.

B.7 ADDITIONAL QUANTITATIVE RESULTS

Additional quantitative results are shown in Table B.5 to B.16.
Table B.6: Test NLL on MNIST. “†” and “‡” has the same meaning as Table 1.

<table>
<thead>
<tr>
<th>Model</th>
<th>NLL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Models without PixelCNN decoder</strong></td>
<td></td>
</tr>
<tr>
<td>ConvHVAE + Lars prior</td>
<td>80.30</td>
</tr>
<tr>
<td>ConvHVAE + VampPrior</td>
<td>79.75</td>
</tr>
<tr>
<td>VAE + IAF† (Kingma et al., 2016)</td>
<td>79.10 ± 0.07</td>
</tr>
<tr>
<td>BIVA‡ (Maaløe et al., 2019)</td>
<td>78.41</td>
</tr>
<tr>
<td>ConvVAE + RNVP $p(z)$, $K = 50$</td>
<td>78.61 ± 0.01</td>
</tr>
<tr>
<td>ResnetVAE + RNVP $p(z)$, $K = 50$</td>
<td>78.49 ± 0.01</td>
</tr>
<tr>
<td><strong>Models with PixelCNN decoder</strong></td>
<td></td>
</tr>
<tr>
<td>VLAE‡ (Chen et al., 2017)</td>
<td>78.53</td>
</tr>
<tr>
<td>PixelVAE† (Gulrajani et al., 2017)</td>
<td>79.02</td>
</tr>
<tr>
<td>PixelHVAE + VampPrior† (Tomczak &amp; Welling, 2018)</td>
<td>78.45</td>
</tr>
<tr>
<td>PixelVAE + RNVP $p(z)$, $K = 50$</td>
<td>78.12 ± 0.04</td>
</tr>
</tbody>
</table>

Table B.7: Test NLL on Omniglot. “†” and “‡” has the same meaning as Table 1.

<table>
<thead>
<tr>
<th>Model</th>
<th>NLL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Models without PixelCNN decoder</strong></td>
<td></td>
</tr>
<tr>
<td>ConvHVAE + Lars prior</td>
<td>97.08</td>
</tr>
<tr>
<td>ConvHVAE + VampPrior</td>
<td>97.56</td>
</tr>
<tr>
<td>BIVA‡ (Maaløe et al., 2019)</td>
<td>91.34</td>
</tr>
<tr>
<td>ConvVAE + RNVP $p(z)$, $K = 50$</td>
<td>93.62 ± 0.02</td>
</tr>
<tr>
<td>ResnetVAE + RNVP $p(z)$, $K = 50$</td>
<td>93.52 ± 0.02</td>
</tr>
<tr>
<td><strong>Models with PixelCNN decoder</strong></td>
<td></td>
</tr>
<tr>
<td>VLAE‡ (Chen et al., 2017)</td>
<td>89.83</td>
</tr>
<tr>
<td>PixelHVAE + VampPrior† (Tomczak &amp; Welling, 2018)</td>
<td>89.76</td>
</tr>
<tr>
<td>PixelVAE + RNVP $p(z)$, $K = 50$</td>
<td>89.60 ± 0.01</td>
</tr>
</tbody>
</table>

Table B.8: Test NLL on FashionMNIST. “†” and “‡” has the same meaning as Table 1.

<table>
<thead>
<tr>
<th>Model</th>
<th>NLL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Models without PixelCNN decoder</strong></td>
<td></td>
</tr>
<tr>
<td>ConvHVAE + Lars prior</td>
<td>225.92</td>
</tr>
<tr>
<td>ConvVAE + RNVP $p(z)$, $K = 50$</td>
<td>224.64 ± 0.01</td>
</tr>
<tr>
<td>ResnetVAE + RNVP $p(z)$, $K = 50$</td>
<td>224.07 ± 0.01</td>
</tr>
<tr>
<td><strong>Models with PixelCNN decoder</strong></td>
<td></td>
</tr>
<tr>
<td>PixelVAE + RNVP $p(z)$, $K = 50$</td>
<td>223.36 ± 0.06</td>
</tr>
</tbody>
</table>

B.8 ADDITIONAL QUALITATIVE RESULTS

Additional qualitative results are shown in Fig. B.1 to B.3.
(a) standard  (b) RealNVP $q(z|x)$  (c) RealNVP $p(z)$

Figure B.1: Samples from ResnetVAE trained on MNIST. The last column of each 10x10 grid shows the training set images, most similar to the second-to-last column in pixel-wise L2 distance.

(a) standard  (b) RealNVP $q(z|x)$  (c) RealNVP $p(z)$

Figure B.2: Samples from ResnetVAE trained on FashionMNIST. The last column of each 10x10 grid shows the training set images, most similar to the second-to-last column in pixel-wise L2 distance.

(a) standard  (b) RealNVP $q(z|x)$  (c) RealNVP $p(z)$

Figure B.3: Samples from ResnetVAE trained on Omniglot. The last column of each 10x10 grid shows the training set images, most similar to the second-to-last column in pixel-wise L2 distance.

B.9 ADDITIONAL RESULTS: ABLATION STUDY AND POSTERIOR OVERLAPPING

Additional results about ablation study and posterior overlapping are shown in Table B.9 to B.15.
Table B.9: Test NLL of different models, with both prior and posterior being Gaussian (“standard”), only posterior being RealNVP ("RNVP $q(z|x)$"), and only prior being RealNVP ("RNVP $p(z)$"). Flow depth $K = 20$.

<table>
<thead>
<tr>
<th>Models</th>
<th>Datasets</th>
<th>StaticMNIST</th>
<th>MNIST</th>
<th>FashionMNIST</th>
<th>Omniglot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>standard</td>
<td>88.84 ± 0.05</td>
<td>84.48 ± 0.03</td>
<td>228.60 ± 0.03</td>
<td>106.42 ± 0.14</td>
</tr>
<tr>
<td>DenseVAE</td>
<td>RNVP $q(z</td>
<td>x)$</td>
<td>86.07 ± 0.11</td>
<td>82.53 ± 0.00</td>
<td>227.79 ± 0.01</td>
</tr>
<tr>
<td></td>
<td>RNVP $p(z)$</td>
<td>84.87 ± 0.05</td>
<td><strong>80.43 ± 0.01</strong></td>
<td><strong>226.11 ± 0.02</strong></td>
<td><strong>102.19 ± 0.12</strong></td>
</tr>
<tr>
<td>ConvVAE</td>
<td>standard</td>
<td>83.63 ± 0.01</td>
<td>82.14 ± 0.00</td>
<td>227.51 ± 0.01</td>
<td>97.87 ± 0.02</td>
</tr>
<tr>
<td></td>
<td>RNVP $q(z</td>
<td>x)$</td>
<td>81.11 ± 0.03</td>
<td>80.09 ± 0.01</td>
<td>226.03 ± 0.00</td>
</tr>
<tr>
<td></td>
<td>RNVP $p(z)$</td>
<td>80.06 ± 0.07</td>
<td><strong>78.67 ± 0.01</strong></td>
<td><strong>224.65 ± 0.01</strong></td>
<td><strong>93.68 ± 0.01</strong></td>
</tr>
<tr>
<td>ResnetVAE</td>
<td>standard</td>
<td>82.95 ± 0.09</td>
<td>81.07 ± 0.03</td>
<td>226.17 ± 0.05</td>
<td>96.99 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>RNVP $q(z</td>
<td>x)$</td>
<td>80.97 ± 0.05</td>
<td>79.53 ± 0.03</td>
<td>225.02 ± 0.01</td>
</tr>
<tr>
<td></td>
<td>RNVP $p(z)$</td>
<td>79.99 ± 0.02</td>
<td><strong>78.58 ± 0.01</strong></td>
<td><strong>224.09 ± 0.01</strong></td>
<td><strong>93.61 ± 0.04</strong></td>
</tr>
<tr>
<td>PixelVAE</td>
<td>standard</td>
<td>79.47 ± 0.02</td>
<td>78.64 ± 0.02</td>
<td>224.22 ± 0.06</td>
<td>89.83 ± 0.04</td>
</tr>
<tr>
<td></td>
<td>RNVP $q(z</td>
<td>x)$</td>
<td>79.09 ± 0.01</td>
<td>78.41 ± 0.01</td>
<td>223.81 ± 0.00</td>
</tr>
<tr>
<td></td>
<td>RNVP $p(z)$</td>
<td><strong>78.92 ± 0.02</strong></td>
<td><strong>78.15 ± 0.04</strong></td>
<td><strong>223.40 ± 0.07</strong></td>
<td><strong>89.61 ± 0.03</strong></td>
</tr>
</tbody>
</table>

Table B.10: Test NLL of ResnetVAE, with only RealNVP posterior ("RNVP $q(z|x)$"), only RealNVP prior ("RNVP $p(z)$"), and both RealNVP prior & posterior ("both"). Flow depth $K = 20$.

| Datasets   | RNVP $q(z|x)$ | RNVP $p(z)$ | both       |
|------------|---------------|--------------|------------|
| StaticMNIST | 80.97 ± 0.05 | 79.99 ± 0.02 | **79.87 ± 0.04** |
| MNIST      | 79.53 ± 0.03 | 78.58 ± 0.01 | **78.56 ± 0.01** |
| FashionMNIST | 225.02 ± 0.01 | 224.09 ± 0.01 | **224.08 ± 0.02** |
| Omniglot   | 94.30 ± 0.02 | **93.61 ± 0.04** | 93.68 ± 0.04 |

Table B.11: Test NLL of ResnetVAE, with RealNVP prior of different flow depth $K$. Even $K = 1$, RealNVP prior improves NLLs by about 1 nat. There is no over-fitting for $K$ up to 50. However, we think over-fitting may happen with sufficiently powerful prior, 50-blocks of RealNVP may simply not be deep enough.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Flow depth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StaticMNIST</td>
</tr>
<tr>
<td>0</td>
<td>82.95 ± 0.09</td>
</tr>
<tr>
<td>1</td>
<td>81.76 ± 0.04</td>
</tr>
<tr>
<td>2</td>
<td>81.30 ± 0.02</td>
</tr>
<tr>
<td>5</td>
<td>80.64 ± 0.06</td>
</tr>
<tr>
<td>10</td>
<td>80.26 ± 0.05</td>
</tr>
<tr>
<td>20</td>
<td>79.99 ± 0.02</td>
</tr>
<tr>
<td>30</td>
<td>79.90 ± 0.05</td>
</tr>
<tr>
<td>50</td>
<td><strong>79.84 ± 0.04</strong></td>
</tr>
</tbody>
</table>
Table B.12: Average test ELBO (“elbo”), reconstruction loss (“recons”), $\mathbb{E}_{p^*}(x)D_{KL}(q_\phi(z|x)||p_\lambda(z))$ (“$kl$”) and $\mathbb{E}_{p^*}(x)D_{KL}(q_\phi(z|x)||p_\theta(z|x))$ (“$kl_{z|x}$”) of various DenseVAE. Flow depth $K = 20$.

| Datasets       | elbo      | recons     | kl  | $kl_{z|x}$ |
|----------------|-----------|------------|-----|------------|
| StaticMNIST    | -94.53    | -65.54     | 29.00 | 5.69       |
| MNIST          | -88.19    | -62.04     | 26.16 | 3.71       |
| FashionMNIST   | -230.81   | -211.71    | 19.11 | 2.22       |
| Omniglot       | -113.59   | -78.25     | 35.33 | 7.17       |

| Datasets       | elbo      | recons     | kl  | $kl_{z|x}$ |
|----------------|-----------|------------|-----|------------|
| StaticMNIST    | -90.78    | -62.55     | 28.23 | 4.71       |
| MNIST          | -85.15    | -58.83     | 26.31 | 2.62       |
| FashionMNIST   | -229.53   | -210.35    | 19.18 | 1.75       |
| Omniglot       | -108.54   | -73.71     | 34.83 | 5.57       |

Table B.13: Average test ELBO (“elbo”), reconstruction loss (“recons”), $\mathbb{E}_{p^*}(x)D_{KL}(q_\phi(z|x)||p_\lambda(z))$ (“$kl$”) and $\mathbb{E}_{p^*}(x)D_{KL}(q_\phi(z|x)||p_\theta(z|x))$ (“$kl_{z|x}$”) of various ConvVAE. Flow depth $K = 20$.

| Datasets       | elbo      | recons     | kl  | $kl_{z|x}$ |
|----------------|-----------|------------|-----|------------|
| StaticMNIST    | -88.87    | -63.35     | 25.52 | 4.00       |
| MNIST          | -82.57    | -55.99     | 26.58 | 2.14       |
| FashionMNIST   | -227.72   | -208.13    | 19.59 | 1.61       |
| Omniglot       | -107.99   | -73.10     | 34.89 | 5.80       |

| Datasets       | elbo      | recons     | kl  | $kl_{z|x}$ |
|----------------|-----------|------------|-----|------------|
| StaticMNIST    | -83.92    | -56.67     | 28.90 | 3.86       |
| MNIST          | -82.46    | -56.08     | 27.09 | 2.00       |
| FashionMNIST   | -226.65   | -204.93    | 21.72 | 2.00       |
| Omniglot       | -98.95    | -59.07     | 39.51 | 4.91       |
Table B.14: Average test ELBO (“\textit{elbo}”), reconstruction loss (“\textit{recons}”), \(\mathbb{E}_{p^\star(x)} D_{KL}(q_\phi(z|x)||p_\lambda(z|x))\) (“\textit{kl}”) and \(\mathbb{E}_{p^\star(x)} D_{KL}(q_\phi(z|x)||p_\theta(z|x))\) (“\textit{kl}_z|_x”) of various ResnetVAE. Flow depth \(K = 20\).

| Datasets | elbo   | recons | kl   | \(kl_{z|x}\) |
|----------|--------|--------|------|-------------|
| StaticMNIST | -87.61 | -60.09 | 27.52 | 4.67        |
| MNIST    | -84.62 | -58.70 | 25.92 | 3.55        |
| FashionMNIST | -228.91 | -208.94 | 19.96 | 2.74        |
| Omniglot | -104.87 | -66.98 | 37.89 | 7.88        |

Table B.15: Average test ELBO (“\textit{elbo}”), reconstruction loss (“\textit{recons}”), \(\mathbb{E}_{p^\star(x)} D_{KL}(q_\phi(z|x)||p_\lambda(z|x))\) (“\textit{kl}”) and \(\mathbb{E}_{p^\star(x)} D_{KL}(q_\phi(z|x)||p_\theta(z|x))\) (“\textit{kl}_z|_x”) of various PixelVAE. Flow depth \(K = 20\).

| Datasets | elbo   | recons | kl   | \(kl_{z|x}\) |
|----------|--------|--------|------|-------------|
| StaticMNIST | -84.72 | -58.10 | 26.63 | 3.75        |
| MNIST    | -81.95 | -56.15 | 25.80 | 2.42        |
| FashionMNIST | -227.16 | -206.61 | 20.54 | 2.14        |
| Omniglot | -100.30 | -63.34 | 36.96 | 6.00        |

| Datasets | elbo   | recons | kl   | \(kl_{z|x}\) |
|----------|--------|--------|------|-------------|
| StaticMNIST | -82.85 | -54.32 | 28.54 | 2.87        |
| MNIST    | -80.34 | -53.64 | 26.70 | 1.76        |
| FashionMNIST | -225.05 | -204.66 | 21.31 | 1.88        |
| Omniglot | -99.60 | -61.21 | 38.39 | 5.99        |
Table B.16: Test NLL of ResnetVAE, with prior trained by: joint training, iterative training, post-hoc training, and standard VAE ("none") as reference. Flow depth $K = 20$.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>joint</th>
<th>iterative</th>
<th>post-hoc</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>StaticMNIST</td>
<td>79.99 ± 0.02</td>
<td>80.63 ± 0.02</td>
<td>80.86 ± 0.04</td>
<td>82.95 ± 0.09</td>
</tr>
<tr>
<td>MNIST</td>
<td>78.58 ± 0.01</td>
<td>79.61 ± 0.01</td>
<td>79.90 ± 0.04</td>
<td>81.07 ± 0.03</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>224.09 ± 0.01</td>
<td>224.88 ± 0.02</td>
<td>225.22 ± 0.01</td>
<td>226.17 ± 0.05</td>
</tr>
<tr>
<td>Omniglot</td>
<td>93.61 ± 0.04</td>
<td>94.43 ± 0.11</td>
<td>94.87 ± 0.05</td>
<td>96.99 ± 0.04</td>
</tr>
</tbody>
</table>

Figure B.4: Avg. std of $q_\phi(z|x)$ of $\beta$-ResnetVAE trained with different $\beta$ and prior flow depth $K$. 
B.10 ADDITIONAL RESULTS: TWO LEVEL HIERARCHICAL VAE WITH FLOW PRIOR

In addition to the comprehensive experiments based on VAE, we also conduct experiments using two-level ResnetHVAE (Tomczak & Welling, 2018), just as the two major previous works (Tomczak & Welling, 2018; Bauer & Mnih, 2019).

Architectures of ResnetHVAE for binarized datasets The observed variable \( x \) is a 3-d tensor, just the same as VAEs. The latent variables \( z_1 \) and \( z_2 \) are vectors with the same number of dimensions, which is 40 on MNIST and StaticMNIST, and 64 on FashionMNIST and Omniglot. We use \( \text{Dim}(z) \) to denote this dimensionality.

The overall factorization for HVAE is:
\[
    p_\theta(x, z_1) = p_\theta(x|z_1, z_2) p_\theta(z_1|z_2) p_\lambda(z_2)
\]
\[
    q_\phi(z_2, z_1|x) = q_\phi(z_1|z_2, x) q_\phi(z_2|x)
\]
where \( p_\lambda(z_2) \) is either a flow prior or a unit Gaussian prior, depending on the context.

The shared feature network \( h_\phi(x) \) for \( q_\phi(z_2|x) \) and \( q_\phi(z_1|x) \) is:
\[
    h_\phi(x) = x \rightarrow \text{Resnet}[28 \times 28 \times 32 \rightarrow 28 \times 28 \times 32 \rightarrow 14 \times 14 \times 64 \rightarrow 14 \times 14 \times 64 \rightarrow 7 \times 7 \times 64 \rightarrow 7 \times 7 \times 16] \rightarrow \text{Flatten}
\]

The Gaussian posterior \( q_\phi(z_2|x) \) is derived as:
\[
    q_\phi(z_2|x) = \mathcal{N}(\mu_\phi(h_\phi(x)), \sigma^2_\phi(h_\phi(x)) \mathbf{I})
\]
\[
    \mu_\phi(h_\phi(x)) = h_\phi(x) \rightarrow \text{Linear}[\text{Dim}(z)]
\]
\[
    \log \sigma_\phi(h_\phi(x)) = h_\phi(x) \rightarrow \text{Linear}[\text{Dim}(z)]
\]

The Gaussian posterior \( q_\phi(z_1|x, z_2) \) is derived as:
\[
    q_\phi(z_1|x, z_2) = \mathcal{N}(\mu_\phi(g_\phi(x, z_2)), \sigma^2_\phi(g_\phi(x, z_2)) \mathbf{I})
\]
\[
    \mu_\phi(g_\phi(x, z_2)) = g_\phi(x, z_2) \rightarrow \text{Linear}[\text{Dim}(z)]
\]
\[
    \log \sigma_\phi(g_\phi(x, z_2)) = g_\phi(x, z_2) \rightarrow \text{Linear}[\text{Dim}(z)]
\]
\[
    g_\phi(x, z_2) = \text{Concat}[t_\phi(x), u_\phi(z_2)] \rightarrow \text{Dense}[300]
\]
\[
    t_\phi(x) = h_\phi(x) \rightarrow \text{Dense}[150 \rightarrow 150]
\]
\[
    u_\phi(z_2) = z_2 \rightarrow \text{Dense}[150 \rightarrow 150]
\]

The Gaussian prior \( p_\theta(z_1|z_2) \) is derived as:
\[
    p_\theta(z_1|z_2) = \mathcal{N}(\mu_\theta(t_\theta(z_2)), \sigma^2_\theta(t_\theta(z_2)) \mathbf{I})
\]
\[
    \mu_\theta(t_\theta(z_2)) = t_\theta(z_2) \rightarrow \text{Linear}[\text{Dim}(z)]
\]
\[
    \log \sigma_\theta(t_\theta(z_2)) = t_\theta(z_2) \rightarrow \text{Linear}[\text{Dim}(z)]
\]
\[
    t_\theta(z_2) = z_2 \rightarrow \text{Dense}[150 \rightarrow 150]
\]

For binarized images, the Bernoulli conditional distribution \( p_\theta(x|z_1, z_2) \) is derived as:
\[
    p_\theta(x|z_1, z_2) = \text{Bernoulli}[\mu_\theta(g_\theta(z_1, z_2))]
\]
\[
    \log \frac{\mu_\theta(g_\theta(z_1, z_2))}{1 - \mu_\theta(g_\theta(z_1, z_2))} = g_\theta(z_1, z_2) \rightarrow \text{LinearConv}_{1 \times 1}[28 \times 28 \times 1]
\]
\[
    g_\theta(z_1, z_2) = \text{Concat}[z_1, z_2] \rightarrow h_\theta(z)
\]
\[
    h_\theta(z) = \text{Dense}[784] \rightarrow \text{UnFlatten}[7 \times 7 \times 16] \rightarrow \text{DeResnet}[7 \times 7 \times 64 \rightarrow 14 \times 14 \times 64 \rightarrow 28 \times 28 \times 32 \rightarrow 28 \times 28 \times 32]
\]

Note the shared feature network \( h_\phi(x) \) and the decoder network \( h_\theta(z) \) have exactly the same architecture as VAE (see Appendix B.2).
Table B.17: Test negative log-likelihood of ResnetHVAE, compared with ResnetVAE. Flow depth $K = 20$ for $p_\lambda(z)$ in ResnetVAE. For ResnetHVAE, $p_\lambda(z_2)$ is a flow prior with $K = 20$, and $p_\lambda(z_1 | x)$ is a Gaussian prior.

<table>
<thead>
<tr>
<th>Models</th>
<th>StaticMNIST</th>
<th>MNIST</th>
<th>FashionMNIST</th>
<th>Omniglot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResnetVAE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standard</td>
<td>82.95 ± 0.09</td>
<td>81.07 ± 0.03</td>
<td>226.17 ± 0.05</td>
<td>96.99 ± 0.04</td>
</tr>
<tr>
<td>RNVP $p(z)$</td>
<td>79.99 ± 0.02</td>
<td>78.58 ± 0.01</td>
<td>224.09 ± 0.01</td>
<td>93.61 ± 0.04</td>
</tr>
<tr>
<td>ResnetHVAE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standard</td>
<td>82.13 ± 0.10</td>
<td>80.19 ± 0.04</td>
<td>225.16 ± 0.04</td>
<td>96.02 ± 0.06</td>
</tr>
<tr>
<td>RNVP $p(z_2)$</td>
<td>80.35 ± 0.05</td>
<td><strong>78.57 ± 0.01</strong></td>
<td>224.10 ± 0.03</td>
<td><strong>93.57 ± 0.05</strong></td>
</tr>
</tbody>
</table>

Table B.18: Average number of active units of ResnetHVAE of $z_2$ and $z_1$, with unit Gaussian $p(z_2)$ and $p(z_1)$ (“standard”), and RealNVP $p(z_2)$ and Gaussian $p(z_1)$ (“RNVP $p(z_2)$”).

<table>
<thead>
<tr>
<th>Datasets</th>
<th>StaticMNIST</th>
<th>MNIST</th>
<th>FashionMNIST</th>
<th>Omniglot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResnetHVAE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standard $z_1$</td>
<td>8.7</td>
<td></td>
<td>10.3</td>
<td>16</td>
</tr>
<tr>
<td>standard $z_2$</td>
<td>22</td>
<td>17</td>
<td>15.7</td>
<td>14</td>
</tr>
<tr>
<td>RealNVP $p(z_2)$</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>64</td>
</tr>
</tbody>
</table>

Experimental results Table B.17 shows the test negative log-likelihood (NLL) of ResnetHVAE, with or without RealNVP $p(z_2)$, compared with ResnetVAE, with or without RealNVP $p(z)$. We can see that, ResnetHVAE without flow prior has significantly better test NLLs than ResnetVAE without flow prior. However, ResnetHVAE with RealNVP $p(z_2)$ only has roughly the same test NLLs as ResnetVAE with RealNVP $p(z)$ (except on StaticMNIST, where ResnetHVAE with RealNVP $p(z_2)$ is worse than ResnetVAE with $p(z)$, which is likely caused by optimization problems, since over-fitting can happen on StaticMNIST more easily than the other three datasets).

Further investigation on the active units of ResnetHVAE (Table B.18) reveals the cause of such experimental results: when coupled with RealNVP $p(z_2)$, the second latent variable $z_1$ in ResnetHVAE degenerates, such that no dimension is active. We also added RealNVP to $z_1$, but the results showed no difference.

However, we find such degeneration may not always happen. We conducted a pilot experiment of ResnetHVAE with RealNVP $p(z_2)$ and Gaussian $p(z_1)$ on Cifar-10. The shape of $z_2$ and $z_1$ are both $8 \times 8 \times 16$. The number of active units of $z_2$ is 1024 (i.e., all dimensions are active), while the number of active units of $z_1$ is 768 (specifically, these 768 dimensions consist 12 channels of $8 \times 8$ feature maps). The test bpd of this experiment is 3.96. This suggests that the zero active units of $z_1$ in Table B.18 is not caused by the RealNVP prior. The resnet network architecture we used in our experiments (on the binarized datasets) may have limited the model capacity, such that the ResnetVAE with RealNVP prior is already very close to the limitation of test NLLs in such condition, where adding an auxiliary latent variable will not help.

We also report the reconstruction loss, the KL divergence $D_{KL}(q_\phi(z_2, z_1 | x) || p_\theta(z_2, z_1))$ and $D_{KL}(q_\phi(z_2, z_1 | x) || p_\theta(z_2, z_1 | x))$ of ResnetHVAE with or without RealNVP $p(z_2)$, which have the same trend as ResnetVAE (Table 6).
Table B.19: Average test ELBO (“elbo”), reconstruction loss (“recons”), \(D_{KL}(q_\phi(z_2, z_1|x))\) of \(p_\theta(z_2, z_1|x)\) (“\(k_1\)”), and \(D_{KL}(q_\phi(z_2|z_1|x))\) of \(p_\theta(z_2|z_1|x)\) (“\(k_{z|x}\)”) of of ResnetHVAEs with different prior.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>standard</th>
<th>RealNVP p(z_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>elbo</td>
<td>recon</td>
</tr>
<tr>
<td>StaticMNIST</td>
<td>-86.64</td>
<td>-59.44</td>
</tr>
<tr>
<td>MNIST</td>
<td>-83.17</td>
<td>-57.20</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>-227.51</td>
<td>-207.45</td>
</tr>
<tr>
<td>Omniglot</td>
<td>-103.43</td>
<td>-65.52</td>
</tr>
</tbody>
</table>

### B.11 Additional Results: Learned Prior on Low Posterior Samples

Rosca et al. [2018] observed that some of the samples from unit Gaussian prior can have low \(q_\phi(z)\) and low visual quality, and they name these samples the “low posterior samples”. Unlike this paper, they tried various approaches to match \(q_\phi(z)\) to unit Gaussian \(p_\lambda(z)\), including adversarial training methods, but still found low posterior samples scattering across the whole prior. We shall see that learning the prior can avoid having high \(p_\lambda(z)\) on low posterior samples, and thus we suggest adopting learned prior as a cheap solution to this problem.

We shall first introduce the algorithm proposed by Rosca et al. [2018], which can be used to obtain low posterior samples (\(z\) samples which have low likelihoods on \(q_\phi(z)\)) from a trained VAE. Their algorithm first samples a large number of \(z\) from the prior \(p_\lambda(z)\), then uses Monte Carlo estimator to evaluate \(q_\phi(z)\) on these \(z\) samples, and finally chooses a certain number of \(z\) with the lowest \(q_\phi(z)\) likelihoods as the low posterior samples. They also sampled one \(x\) from \(p_\theta(x|z)\) for each low posterior sample \(z\), and plotted the sample means of these \(x\) and the histograms of ELBO on these \(x\). Although we have found their Monte Carlo estimator for \(D_{KL}(q_\phi(z)||p_\lambda(z))\) vulnerable (see Appendix [B.3]), their low posterior samples algorithm only ranks \(q_\phi(z)\) for each \(z\), and the visual results of their algorithm seem plausible. Thus we think this algorithm should be still convincing enough, and we also use it to obtain low posterior samples.

To compare the learned prior with unit Gaussian prior on such low posterior samples, we first train a ResnetVAE with unit Gaussian prior (denoted as standard ResnetVAE), and then add a post-hoc trained RealNVP prior upon this original ResnetVAE (denoted as post-hoc trained ResnetVAE). We then obtain 10,000 \(z\) samples from the standard ResnetVAE, and choose 100 \(z\) with the lowest \(q_\phi(z)\) among these 10,000 samples, evaluated on standard ResnetVAE. Fixing these 100 \(z\) samples, we plot the histograms of \(\log p_\lambda(z)\) w.r.t. standard ResnetVAE and post-hoc trained ResnetVAE. We also obtain one \(x\) sample from \(p_\theta(x|z)\) for each \(z\), and plot the histograms of ELBO for each \(x\) (w.r.t. the two models) and their sample means. See Fig. [B.5]

The \(x\) samples of post-hoc trained ResnetVAE (bottom right) are the same as those of standard ResnetVAE (top right), since post-hoc trained ResnetVAE has exactly the same \(p_\theta(x|z)\) and \(q_\phi(z|x)\) as standard ResnetVAE. However, the post-hoc trained prior successfully assigns much lower \(\log p_\lambda(z)\) (bottom left) than unit Gaussian prior (top left) on the low posterior samples, which suggests that a post-hoc trained prior can avoid granting high likelihoods to these samples in the latent space. Note post-hoc trained ResnetVAE also assigns slightly lower ELBO (bottom middle) than standard ResnetVAE (top middle) to \(x\) samples corresponding to these low posterior samples.

To verify whether a learned prior can avoid obtaining low posterior samples in the first place, we obtained low posterior samples from a ResnetVAE with jointly trained prior (denoted as jointly trained ResnetVAE), see Fig. [B.6]. Compared with Fig. [B.5], we can see that \(\log p_\lambda(z)\) of these low posterior samples and ELBO of the corresponding \(x\) samples are indeed substantially higher than those of standard ResnetVAE and post-hoc trained ResnetVAE. However, the visual quality is not perfect, indicating there is still room for improvement.
Figure B.5: (left) Histograms of $\log p_{\lambda}(z)$ of the low posterior samples, (middle) histograms of ELBO of $x$ samples corresponding to each low posterior sample, and (right) the means of these $x$, on standard ResnetVAE and post-hoc trained ResnetVAE.

Figure B.6: (left) Histograms of $\log p_{\lambda}(z)$ of the low posterior samples, (middle) histograms of ELBO of $x$ samples corresponding to each low posterior sample, and (right) the means of these $x$, on jointly trained ResnetVAE.

B.12 WALL-CLOCK TIMES

Figure B.7: Average training time of various models on MNIST, flow depth $K = 20$. Black sticks are standard deviation bars. For PixelVAE, training mostly terminates in half way due to early-stopping.
We report the average training time of various models trained on MNIST, see Figs. B.7 and B.8. Each experiment runs on one GTX 1080 Ti graphical card. In Fig. B.7, we can see that the computational cost of RealNVP prior is independent with the architecture of other parts of a model. For complicated architectures like ResnetVAE and PixelVAE, the cost of adding a RealNVP prior is fairly acceptable, since it can bring large improvement. In Fig B.8 we can see that for $K > 50$, there is likely to be little gain in test NLL, but the computation time will grow even larger. That’s why we do not try larger $K$ in our experiments.

REFERENCES


