A. KL terms for A and $\boldsymbol{\nu}$

The computation for the KL terms for $\mathbf{A}$ and $\boldsymbol{\nu}$ in Equation 7 is omitted in the main paper and we present here to show how to compute them.

The KL term for $\mathbf{A}$ is the KL between two Normal distributions $q_{\mathbf{A}} = \mathcal{N}(\mu_\phi(\mathbf{x}), \sigma_\phi(\mathbf{x}))$ and $p_{\mathbf{A}} = \mathcal{N}(0, I)$. We use its closed-form expression adapted from the Appendix of Kingma & Welling (2013)

$$\mathrm{KL}\left[q(\mathbf{A}) \,\|\, p(\mathbf{A})\right] = -\frac{1}{2}\sum_{k=1}^{\infty}\left(1 + \log((\sigma_k)^2) - (\mu_k)^2 - \sigma_k)^2\right), \tag{19}$$

where $\mu_k := \mu_{\phi_k}(\mathbf{x})$ and $\sigma_k := \sigma_{\phi_k}(\mathbf{x})$.

The KL term for $\boldsymbol{\nu}$ is the KL between the Kumaraswamy and Beta distributions $q(\nu_k) = \mathrm{Kumaraswamy}(a_k, b_k)$ and $p(\nu_k) = \mathrm{Beta}(\alpha, 1)$. We use its closed-form expression adapted from Appendix of Nalisnick & Smyth (2017)

$$\mathrm{KL}\left[q(\boldsymbol{\nu}) \,\|\, p(\boldsymbol{\nu})\right] = \sum_{k=1}^{\infty}\mathrm{KL}\left[\mathrm{Kumaraswamy}(a_k, b_k) \,\|\, \mathrm{Beta}(\alpha, 1)\right], \tag{20}$$

where

$$\begin{aligned}&\mathrm{KL}\left[\mathrm{Kumaraswamy}(a, b) \,\|\, \mathrm{Beta}(\alpha, \beta)\right]\\ =&\frac{a-\alpha}{a}\left(-\gamma - \Psi(b) - \frac{1}{b}\right) + \log ab + \log B(\alpha, \beta) - \frac{b-1}{b} + (\beta - 1)b\sum_{m=1}^{\infty}\frac{1}{m+ab}B\left(\frac{m}{a}, b\right).\end{aligned} \tag{21}$$

We approximate the infinite sum in Equation 21 using its first 11 terms as it is suggested by Nalisnick & Smyth (2017).

B. Proof of the Inequality in Equation 9

Equation 9 states the inequality that

$$q(\mathbf{Z}|\boldsymbol{\nu}) = \sum_{j=1}^{\infty} m_j q(\mathbf{Z}|K^* = j, \boldsymbol{\nu}) \le q(\mathbf{Z}|K^* = K^\dagger, \boldsymbol{\nu}), \tag{9}$$

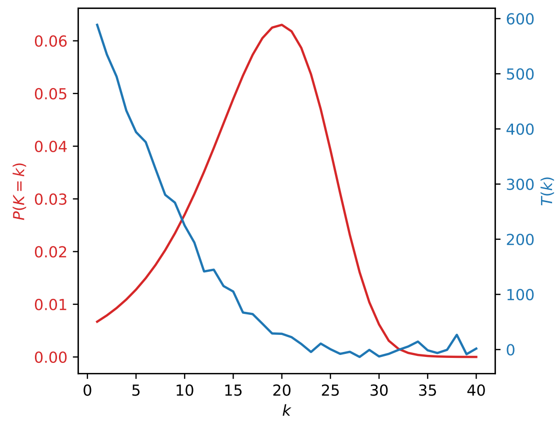
where $K^\dagger := \max_k\{\exists n, z_{nk} \neq 0\}$, the maximum column index for which that column of $\mathbf{Z}$ is not all 0s.

To prove this, we begin with the definition of $q_{\mathbf{Z}}$ given a truncation level j

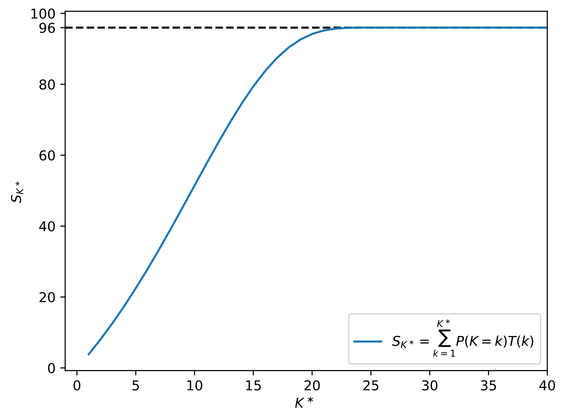
$$q(\mathbf{Z}|K^* = j, \boldsymbol{\nu}) := \delta\{j \ge K^\dagger\}\prod_{n=1}^{N}\prod_{k=1}^{j}\pi_k^{z_{nk}}(1-\pi_k)^{(1-z_{nk})}. \tag{22}$$

First, $q(\mathbf{Z}|K^* = j, \boldsymbol{\nu}) = 0$ for $j < K^\dagger$ because of the delta function that comes from the truncation. Second, $q(\mathbf{Z}^k|K^* = j, \boldsymbol{\nu})$ is a monotonically decreasing function for $j \ge K^\dagger$. To see this, consider $q(\mathbf{Z}|K^* = l, \boldsymbol{\nu})$ for which $K^\dagger \le j < l$

$$\begin{aligned}q(\mathbf{Z}|K^* = l, \boldsymbol{\nu}) &= \delta\{l \ge K^\dagger\}\prod_{n=1}^{N}\prod_{k=1}^{l}\pi_k^{z_{nk}}(1-\pi_k)^{(1-z_{nk})}\\ &= \prod_{n=1}^{N}\left(\prod_{k=1}^{j}\pi_k^{z_{nk}}(1-\pi_k)^{(1-z_{nk})}\prod_{k=j+1}^{l}\pi_k^{z_{nk}}(1-\pi_k)^{(1-z_{nk})}\right)\\ &= \left(\prod_{n=1}^{N}\prod_{k=1}^{j}\pi_k^{z_{nk}}(1-\pi_k)^{(1-z_{nk})}\right)\left(\prod_{n=1}^{N}\prod_{k=j+1}^{l}\pi_k^{z_{nk}}(1-\pi_k)^{(1-z_{nk})}\right)\\ &= q(\mathbf{Z}|K^* = j, \boldsymbol{\nu})\left(\prod_{n=1}^{N}\prod_{k=j+1}^{l}\pi_k^{z_{nk}}(1-\pi_k)^{(1-z_{nk})}\right)\\ &=: q(\mathbf{Z}|K^* = j, \boldsymbol{\nu})Q\\ &\le q(\mathbf{Z}|K^* = j, \boldsymbol{\nu})\end{aligned} \tag{23}$$



(a) $P(K=k)$ and $T(k)$



(b) S_{K^*}

Figure 5. Functions and the target summation in the toy example

The last step comes from the fact that $Q <= 1$ since $\pi_k \in [0,1], \forall k = 1, \ldots, \infty$. Now we can show

$$
\begin{aligned}
q(\mathbf{Z}|\boldsymbol{\nu}) &= \sum_{j=1}^{\infty} m_j q(\mathbf{Z}|K^* = j, \boldsymbol{\nu}) \\
&= \sum_{j=1}^{K^\dagger - 1} m_j q(\mathbf{Z}|K^* = j, \boldsymbol{\nu}) + \sum_{j=K^\dagger}^{\infty} m_j q(\mathbf{Z}|K^* = j, \boldsymbol{\nu}) \\
&\leq \sum_{j=K^\dagger}^{\infty} m_j q(\mathbf{Z}|K^* = K^\dagger, \boldsymbol{\nu}) \\
&\leq \sum_{j=1}^{\infty} m_j q(\mathbf{Z}|K^* = K^\dagger, \boldsymbol{\nu}) = q(\mathbf{Z}|K^* = K^\dagger, \boldsymbol{\nu})
\end{aligned}
$$

C. Comparison of Russian roulette estimation against naive Monte Carlo estimation

As a way to motivate the Russian roulette sampler, in this section we show the empirical evidence that, when estimating an infinite summation like (13) or (16), naive Monte Carlo can have very high variance which cannot be easily overcame by using more MC samples.

Suppose that we want to estimate $S = \lim_{K^* \to \infty} S_{K^*}$ where $S_{K^*} = \sum_{k=1}^{K^*} P(K = k)T(k)$. In this illustration, we set $T(k) = T^*(k) + \mathcal{N}(0,1)$ where

$$
T^*(k) = \begin{cases} (k-25)^2 & k < 25 \\ 0.01(k-25) & k \geq 25 \end{cases}.
$$

in order to mimic a common loss curve for different sized truncation levels, and set $P(K = k) = m_k = (1 - \rho_{k+1}) \prod_{i=1}^{k} \rho_i$ where

$$
\rho_k = \begin{cases} \frac{1}{1+\exp(-5(k-1)/29)} & k < 30 \\ 0.5 & k \geq 30 \end{cases}.
$$

in order to mimic a distribution of truncation level which is still away from an optimal one. Figure 5 shows $T(k)$ and $P(K = k)$ (Figure 5(a)) as well as how S_{K^*} changes with different level of K^* (Figure 5(b)). As can be seen from Figure 5(b), S is approximately 96.

We compare to the native Monte Carlo estimate in which we sample $k_1 \dots k_N$ independently from $P(K)$ and compute

$$\hat{S}_{MC} = \frac{1}{N} \sum_{i=1}^{N} T(k_i).$$

We run each estimation for 100 times with the number of samples varying from 1 to 200. We report the mean and variance over the 100 runs. We also report the *efficiency* (Pharr et al., 2016) which is defined as $1/vt$ where v is the variance and t is the time required by the estimation. These results are given in Figure 6. As you can see from the top plot for the mean, both estimates are unbiased towards the approximated true value. However, the variance of the naive Monte Carlo estimator is much higher. This is shown more clearly in the plot in the middle, which shows how the variance asymptotically changes with the number of samples. It can be seen that the Russian roulette estimates have much lower variance than naive Monte Carlo. In fact, the naive Monte Carlo estimation retains a variance of 63.5 even with 200 samples while Russian roulette reaches a variance of around 10 with 15 samples. The final plot in the bottom compares the efficiency of the two estimators and it can be seen that the efficiency for the naive Monte Carlo one increases very slowly with more samples. Thus we can conclude that using a naive Monte Carlo estimation for an infinite summation like (13) or (16) is of very high variance and the variance cannot be easily overcame by using more MC samples.

D. Algorithmic description for effective RAVE

A naive implementation requires running the encoder and decoder once for each Russian roulette sample, but in this appendix we show how computation can be reused across samples. When we average over M independent Russian roulette samples, of each the truncation level is τ_m, the gradient estimate for ρ_k becomes

$$\hat{\partial}^M_{\rho_k} := \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{\tau_m} a_i T_i, . \tag{24}$$

where $T_i = \tilde{\mathcal{L}}^i$ and we define

$$a_i = (1 - \rho_{i+1}) w_i. \tag{25}$$

A naive way to compute the weighted summation of $\hat{\partial}^M_{\rho_k}$ is by running the encoder and decoder multiple times for each truncation level τ_m required and sum up, but this involves much wasted computation.

Instead, a re-weighting trick be applied to reuse computation. We first draw M samples $\{\tau_m\}_{m=1}^{M}$ of the truncation level from the distribution defined by $P(\tau_m = k) = m_k$. Then we compute $\tau^* = \max_m \tau_m$. We run the encoder once at the truncation level of $K^* = \tau^*$ and run the decoder multiple times (with truncation level $K^* = 1, \dots, \tau^*$), and compute the final estimate as the weighted sum

$$\hat{\partial}^M_{\rho_k} = \sum_{i=1}^{\tau^*} b_i T_i, \tag{26}$$

where

$$b_i = \frac{1}{M} \sum_{m=1}^{M} a_i \delta(\tau_m \geq i). \tag{27}$$

It can be seen by reordering terms that this is equal to (24).

A similar re-weighting method applies to computing $\hat{\partial}^M_{\psi_k}$ with M samples, for which one only needs to re-weight each term in the forward pass of the EBLO computation as

$$\tilde{\mathcal{L}}^M = \sum_{i=1}^{\tau^*} c_i T_i, \text{ where } c_i = \frac{1}{M} \sum_{m=1}^{M} (1 - \rho_{i+1}) \delta(\tau_m \geq i) \tag{28}$$

and automatic differentiation can compute the gradient for all the parameters in the inference and generative network in (14).

Note that for the KL term, $\mathcal{K}_{1:i}$, in each $\tilde{\mathcal{L}}^i$ for $i = 1, \dots, \tau^*$, as $\mathcal{K}_{1:i}$ is a sum of independent KL terms for each feature, $\mathcal{K}_j$, this term can be computed after the single run of encoder by $\mathcal{K}_{1:i} = \sum_{j=1}^{i} \mathcal{K}_j$, where $\mathcal{K}_j$ is the KL term of the j-th feature.

The complete algorithm that uses this re-weighting trick is given in Algorithm 2. This requires a single run of the encoder and τ^* runs of the decoder. Additionally, Equation (26) for all ks (i.e. Line 10 in Algorithm 2) can be vectorized and implemented as matrix multiplications. We omit the details here and refer readers to our source code for concrete vectorization.

Algorithm 2 Effective RAVE with re-weighting trick.

```
input $\{\mathbf{X}_i\}_{i=1}^{B}$: $B$ mini batches of data
input $M$: the number of Russian roulette samples to use
 1: for $i = 1, \dots, B$ do
 2:    Sample $M$ samples $\{\tau_m\}_{m=1}^{M}$ from the truncation distribution $m_k$
 3:    Compute $\tau^* = \max_m \tau_m$
 4:    Compute $\{b_k\}_{k=1}^{\tau^*}$ following (27)
 5:    Encode $\mathbf{X}_i$ into variational distributions with a truncation level of $\tau^*$
 6:    Compute KL between variational posterior and prior for each level $\{\mathcal{K}_{1:k}\}_{k=1}^{\tau^*}$
 7:    for $k = 1, \dots, \tau^*$ do
 8:       Compute the expected reconstruction term $\mathcal{R}_k$ in $T_k$ under variational distributions
 9:    Compute ELBO $\tilde{\mathcal{L}}^i$ for each level $i$: $\{\tilde{\mathcal{L}}^i = \mathcal{R}_k - \mathcal{K}_{1:k}\}_{k=1}^{\tau^*}$
10:    Compute $\hat{\partial}_{\rho_k}^{M}$ for $k = 1, \dots, \tau^*$ using (26)
11:    Compute and return the weighted ELBO $\tilde{\mathcal{L}}^M$ in (28) to automatic differentiation
12:    Obtain $\{\hat{\partial}_{\psi_k}^{M}\}_{k=1}^{\tau^*}$ from automatic differentiation
13:    Update $\{\rho_k, \psi_k\}_{k=1}^{\tau^*}$ using $\{\hat{\partial}_{\rho_k}^{M}, \hat{\partial}_{\psi_k}^{M}\}_{k=1}^{\tau^*}$ via gradient optimization methods
```

E. Training details

For optimization, we use Adam (Kingma & Ba, 2014) with a learning rate of 0.001 and momentum parameters set to 0.99 and 0.999, for all parameters except for ρ. For ρ we use a stochastic gradient descent optimizer (Robbins & Monro, 1985) with a learning rate of 0.002. During training, the temperature of Concrete reparameterization is set to 0.1. We found that in order to to use a low temperature, it is necessary to use high-precision 64-bit floating-point numbers. Using 32-bit floating point numbers with a temperature of 0.1 frequently results in numerical errors.

Note again that we use a multiplier on the KL term for $\boldsymbol{\nu}$ for structured variational methods during training to encourage adhering to the IBP prior, following Singh et al. (2017). It is set to 1,000 on both MNIST and FMNIST datasets. This has a similar effect of using a large α in the IBP prior.

F. Visualization of component collapsing of S-IBP on Fashion-MNIST

For Fashion-MNIST, the corresponding component collapsing visualization of Figure 4 is given in Figure 7. Note that worse than MNIST, there are even dummy features that are activated as frequently as active features, e.g. the 7-th features in Figure 7(a) is shown to be frequently activated in Figure 7. The number of active features shown in Figure 7(a) is 8, which is the mode of the truncation distribution inferred by RRS-IBP, indicated in vertical red lines in both plots, which means RRS-IBP successfully inferred the same number of active features.

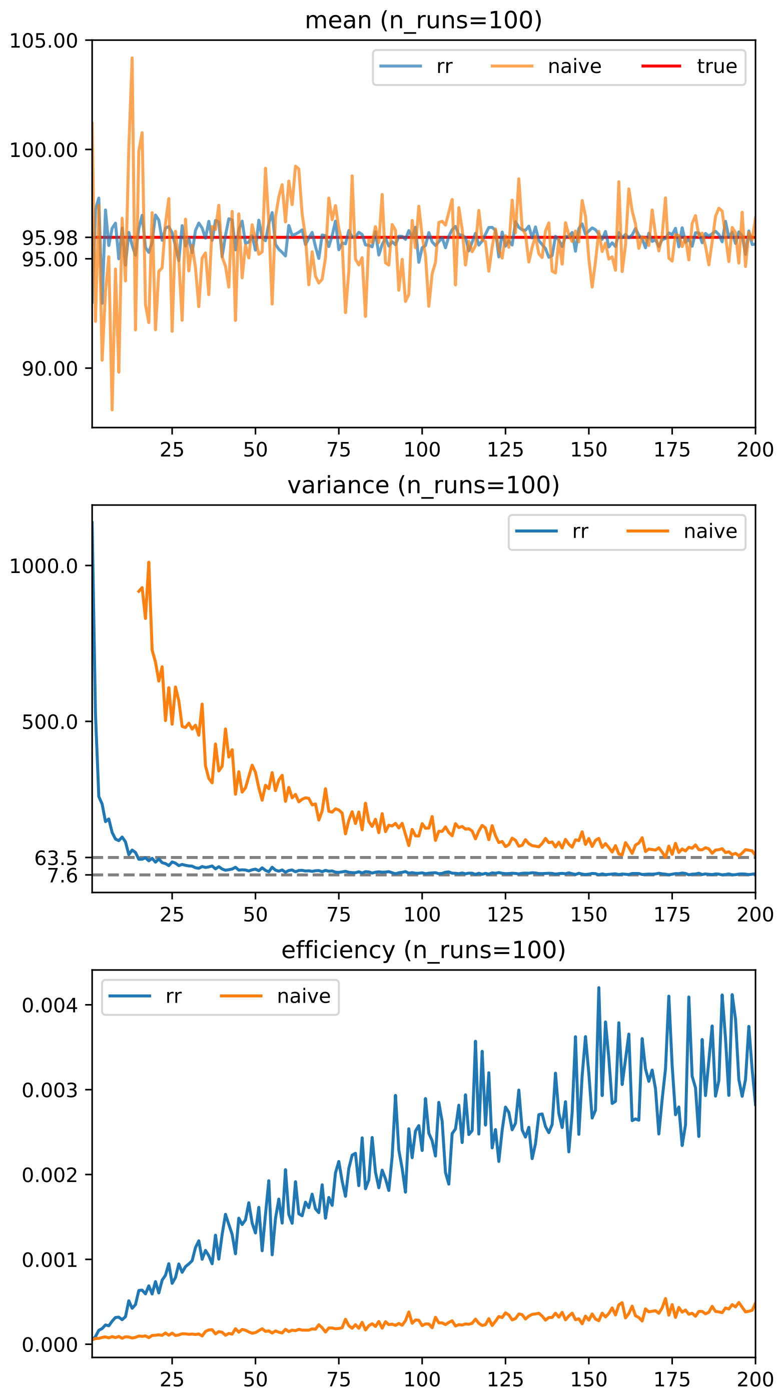
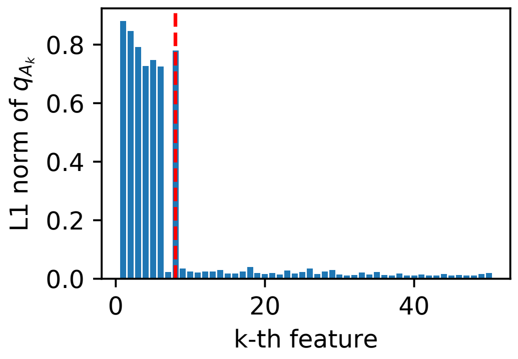
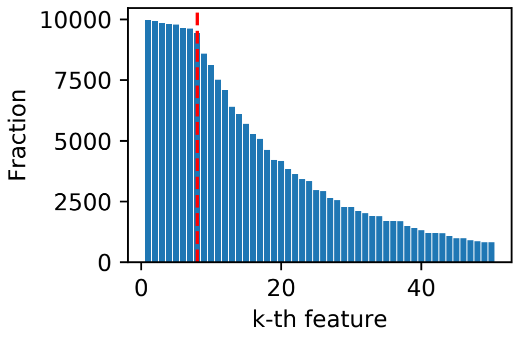


Figure 6. Mean, variance and efficiency for Russian roulette and naive Monte Carlo estimation. In all plots, `rr` stands for Russian roulette and `naive` stands for naive Monte Carlo. In the first plot, `true` represents the value of S that we are estimating.



(a) Mean absolute value of $q_{\mathbf{A}}$



(b) Activation frequency of $\mathbf{Z}$

Figure 7. Effects of truncation level for S-IBP with a deep decoder. All plots are computed from the whole testing set of Fashion-MNIST dataset. The vertical line at 8.0 in red is the corresponding truncation level learned by RRS-IBP.