A. Asymmetric Numeral Systems (ANS)

We will describe a version of Assymetric Numeral Systems that we have assumed access to throughout the paper and used in the experiments, namely the range variant (rANS). All the other versions and interpretations can be found in (Duda, 2009).

ANS encodes a (sequence of) data point(s) into a natural number $s \in \mathbb{N}$, which is called the *state*. We will use unconventional notation, yet consistent with our work: x to denote a single datapoint and s to denote the state. The goal is to obtain a state s whose length of the *binary* representation grows with a rate that closely matches the *entropy* of the data distribution involved.

Suppose we wish to encode a datapoint x that can take on two symbols $\{x_1, x_2\}$ that have equal probability. Starting with $s = 1$. A valid scheme for this distribution is

$$\begin{aligned} x_1 &: s \to 2s \\ x_2 &: s \to 2s + 1. \end{aligned} \tag{11}$$

This simply assigns 0 to x_0 and 1 to x_1 in binary. Therefore, it appends 1 or 0 to the right of the binary representation of the state s. Note that this scheme is fully decodable: if the current state is s', we can read of the last encoded symbol by telling if the state s' is even (last symbol was x_1) or odd (last symbol was x_2). Consequently, after figuring out which symbol x was last encoded, the state s before encoding that symbol is obtained by

$$\begin{aligned} x_1 \ (s' \text{ even}) &: s \to \frac{s'}{2} \\ x_2 \ (s' \text{ odd}) &: s \to \frac{s'-1}{2}. \end{aligned} \tag{12}$$

Now, for the general case, suppose that the datapoint x can take on a multitude of symbols $x \in \{x_1 = 1, x_2 = 2, ..., x_I = I\}$ with probability $\{p_1, p_2, ..., p_I\}$. In order to obtain a scheme analogous to the case with two symbols $\{x_1, x_2\}$, we have to assign every possible symbol x_i to a specific *subset of the natural numbers* $\mathcal{S}_i \subset \mathbb{N}$, that *partitions* the natural numbers. Consequently, $\mathbb{N}$ is a disjoint union of the subsets S_i. Also, the elements in the subset $s \in S_i$ corresponding to x_i must be chosen such that they occur in $\mathbb{N}$ with probability p_i.

This is accomplished by choosing a multiplier M, called the precision of ANS, that scales up the probabilities $\{p_1, p_2, ..., p_I\}$. The scaled up probability p_i is denoted by $F[i]$ and the F's are chosen such that $\sum_{i=1}^{I} F[i] = M$. We also choose subsets $\{K_1, K_2, ...\}$ that form intervals of length M and partition the natural numbers. That means, the first M numbers belong to K_1, the second M numbers belong to K_2, and so on. Then, in every partition K_n, the first Mp_1 numbers are assigned to symbol x_1 and form the subset S_{n1}, the second Mp_2 numbers are assigned to symbol x_2 and form the subset S_{n2}, and so on.

Now, we define $\mathcal{S}_i = \cup_{n=1}^{\infty} S_{ni}$. The resulting subsets $\mathcal{S}_i$ partition the natural numbers $\mathbb{N}$. Furthermore, the elements of $\mathcal{S}_i$ occur with probability approximately equal to p_i in $\mathbb{N}$.

Now, suppose we are given an initial state s. The scheme rANS can be interpreted as follows. **Encoding** a symbol x_i is done by converting the state s to a new state s' that equals the s^{th} occurrence in the set $\mathcal{S}_i$. This operation is made concrete in the following formula:

$$\begin{aligned} C(x, s) &= M \left\lfloor \frac{s}{F[x]} \right\rfloor + B[x] + s \pmod{F[x]} \\ &= s' \end{aligned} \tag{13}$$

where $B[x] = \sum_{i=1}^{x-1} F[i]$, $R = s' \pmod{M}$ and $\lfloor \, \rfloor$ denotes the floor function.

Furthermore, suppose we are given a state s' and we wish to know which number was last encoded (or in other words, we wish to **decode** from s'). Note that the union of the subsets $\mathcal{S}_i$ partitions the the natural numbers $\mathbb{N}$, so every number can be uniquely identified with one of the symbols x_i. Afterwards, if we know what the last encoded symbol x_i was, we can figure out the state s that *preceded* that symbol by doing a look-up for x_i in the set $\mathcal{S}_i$. The index of x_i in $\mathcal{S}_i$ equals the state s that preceded s'. This operation is made concrete in the following formula, which returns a symbol-state (x, s) pair.

$$\begin{aligned} D(s') &= \left(\operatorname{argmax}\{B[x] < R\}, F[x] \lfloor \frac{s'}{M} \rfloor + R - B[x] \right) \\ &= (x, s) \end{aligned} \tag{14}$$

The new state s' after encoding x_i using this scheme is approximately equal to $\frac{s}{p_i}$. Consequently, encoding a sequence of symbols $x_1, x_2, ..., x_T$ onto the initial state s results in a state s_T approximately equal to $s_T \approx \frac{s}{p_1 p_2 \cdots p_T}$. Thus, the resulting codelength is

$$\log s_T \approx \log s + \sum_t \log \frac{1}{p_T} \tag{15}$$

If we divide by T, we obtain an average codelength which approaches the entropy of the data.

B. The bits-back argument

We will present a detailed explanation of the bits-back argument that is fitted to our case. Again, suppose that a sender would like to communicate a sample $\mathbf{x}$ to a receiver through a code that comprises the minimum amount of bits on average over p_{data}. Now suppose that both the sender and receiver have access to the distributions $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z})$, which parameters are optimized such that $p_{\boldsymbol{\theta}}(\mathbf{x})$ $\left(= \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}\right)$ approximates $p_{\text{data}}(\mathbf{x})$ well. Furthermore, both the sender and receiver have access to an entropy encoding technique like ANS. A naive compression scheme for the **sender** is

1. Sample $\mathbf{z} \sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$
2. Encode $\mathbf{x}$ using $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
3. Encode $\mathbf{z}$ using $p(\mathbf{z})$.

This would result in a bitrate equal to $N_{\text{total}} = -\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z})$. The resulting bitstream gets sent over and the **receiver** would proceed in the following way:

1. Decode $\mathbf{z}$ using $p(\mathbf{z})$
2. Decode $\mathbf{x}$ using $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

Consequently, the sample $\mathbf{x}$ is recovered in a lossless manner at the receiver's end. However, we can do better using the bits-back argument, which lets us achieve a bitrate of $\log q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z})$, which is equal to $N_{\text{total}} - \log q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. To understand this, we have to clarify what *decoding* means. If the model fits the true distributions perfectly, entropy encoding can be understood as a (bijective) mapping between datapoints and uniformly random distributed bits. Therefore, assuming that the bits are uniformly random distributed, decoding information $\mathbf{x}$ or $\mathbf{z}$ from the bitstream can be interpreted as sampling $\mathbf{x}$ or $\mathbf{z}$.

Now, assume the sender has access to an arbitrary bitstream N_{init} that is already set in place. This can be in the form of previously compressed datapoints or other auxilary information. Looking at the naive compression scheme, we note that the step 'Sample $\mathbf{z} \sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$' can be substituted by 'Decode $\mathbf{z}$ using $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$'. Consequently, the compression scheme at the **sender**'s end using the bits-back argument is

1. Decode $\mathbf{z}$ using $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$
2. Encode $\mathbf{x}$ using $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
3. Encode $\mathbf{z}$ using $p(\mathbf{z})$.

This results in a total bitrate equal to $N_{\text{total}} = N_{\text{init}} + \log q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z})$. The *total* resulting bitstream gets sent over, and the **receiver** now proceeds as:

1. Decode $\mathbf{z}$ using $p(\mathbf{z})$.
2. Decode $\mathbf{x}$ using $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
3. Encode $\mathbf{z}$ using $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$

and, again, the sample $\mathbf{x}$ is recovered in a lossless manner. But now, in the last step, the receiver has recovered the N_{init} bits of auxiliary information that were set in place, thus gaining N_{init} bits "back". Ignoring the initial bits N_{init}, the *net* number of bits regarding $\mathbf{x}$ is

$$N_{\text{total}} - N_{\text{init}} = \log q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z}) \quad (16)$$

which is on average equal to the negative ELBO $-\mathcal{L}(\boldsymbol{\theta})$.

If the N_{init} bits consist of relevant information, the receiver can then proceed by decompressing that information after gaining these "bits back". As an example, Townsend et al. (2019) point out that it is possible to compress a sequence of datapoints $\{\mathbf{x}_{1:N}\}$, where every datapoint $\mathbf{x}_i$ (except for the first one $\mathbf{x}_1$) uses the bitstream built up thus far as initial bitstream. Then, at the receiver's end, the datapoints $\{\mathbf{x}_{1:N}\}$ get decompressed in reverse order. This way the receiver effectively gains the "bits back" after finishing the three decompression steps of each datapoint $\mathbf{x}_i$, such that decompression of the next datapoint $\mathbf{x}_{i-1}$ can proceed. The only bits that can be irrelevant or redundant are the initial bits needed to compress the first datapoint $\mathbf{x}_1$, though this information gets amortized when compressing multiple datapoints.

C. AC and ANS: Queue vs. Stack

Entropy encoding techniques make lossless compression possible given an arbitrary discrete distribution $p(\mathbf{x})$ over data. There exist several practical compression schemes, of which the most common flavors are

1. Arithmetic Coding (AC) (Witten et al., 1987)
2. Asymmetric Numeral Systems (ANS). (Duda, 2009)

Both schemes operate by encoding the data into single number (in its binary representation equivalent to a sequence of bits or *bitstream*) and decoding the other way around, where the single number serves as the message to be sent. By doing so, both schemes result in a message length with a small overhead of around 2 bits. That is, in case of compressing $\mathbf{x}$, sending/receiving a corresponding codelength of approximately

$$\mathbb{E}\left[-\log p(\mathbf{x})\right] + 2 \text{ bits}. \quad (17)$$

However, AC and ANS differ in how they operate on the bitstream. In AC, the bitstream is treated as a *queue* structure, where the first bits of the bitstream are the first to be decoded (FIFO).

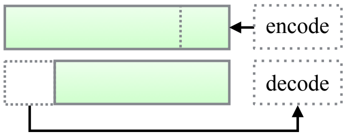


Figure 7: Arithmetic Coding (AC) operates on a bitstream in a *queue*-like manner. Symbols are decoded in the same order as they were encoded.

In ANS, the bitstream is treated as a *stack* structure, where the last bits of the bitstream are the first to be decoded (LIFO).

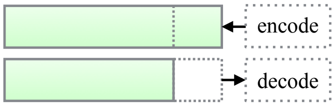


Figure 8: Asymmetric Numeral Systems (ANS) operates on a bitstream in a *stack*-like manner. Symbols are decoded in opposite order as they were encoded.

AC and ANS both operate on one discrete symbol x_d at the time. Therefore, the compression schemes are constrained to a distribution $p(\mathbf{x})$ that encompasses a product of discrete directed univariate distributions in order to operate. Furthermore, both the sender and receiver need to have access to the compression scheme *and* the model in order to be able to communicate.

Frey & Hinton (1996) implement the bits-back theory on a single datapoint $\mathbf{x}$ using AC. Then the *net codelength* is equal to the ELBO, given the fact that we have access to an initial random bitstream. However, we must consider the length of the initial bitstream, which we call the *initial bits*, from which we decode $\mathbf{z}$ when calculating the *actual codelength*, in which case the codelength degenerates to $\mathbb{E}_{q_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}[-\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})]$ bits. So implementing bits-back on a single datapoint will not result in the advantage of getting "bits back".

By communicating a sequence of datapoints $\mathcal{D}$, only the first datapoint $\mathbf{x}^1$ needs to have an initial random bitstream set in place. Afterwards, a subsequent datapoint $\mathbf{x}^i$ may just use the existing bitstream build up thus far to decode the corresponding latent variable $\mathbf{z}^i$. This procedure was first described by (Frey, 1998), and was called bits-back with feedback. We will use the shorter and more convenient term *chaining*, which was introduced by (Townsend et al., 2019).

Chaining is not directly possible with AC, because the existing bitstream is treated as a *queue* structure. Whereas bits-back only works if the latent variable $\mathbf{z} \sim p(\mathbf{z})$ is decoded earlier than the corresponding datapoint $\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, demanding $\mathbf{z}$ to be 'stacked' on top of $\mathbf{x}$ when decoding.

Table 7: Hyperparameters of the model architecture of MNIST, CIFAR-10 and ImageNet (32 × 32). The first three rows denote the dimensions of $\mathbf{x}$, $\mathbf{z}_i$ and the output of the used Residual blocks respectively. The fourth row marks the amount of latent layers L used. The fifth and sixth row denote the amount of 'processing' Residual blocks P and the 'ordinary' Residual blocks B respectively, as explained in D

	MNIST	**CIFAR-10**	**ImageNet** (32×32)
Dimension $\mathbf{x}$ (C, H, W)	$(1, 28, 28)$	$(3, 32, 32)$	$(3, 32, 32)$
Dimension $\mathbf{z}_i$ (C, H, W)	$(1, 16, 16)$	$(8, 16, 16)$	$(8, 16, 16)$
Dimension Residual (C, H, W)	$(64, 16, 16)$	$(256, 16, 16)$	$(256, 16, 16)$
Latent Layers (L)	$\{1, 2, 4, 8\}$	$\{1, 2, 4, 8\}$	$\{1, 2, 4\}$
'Processing' Residual (P)	4	4	4
'Ordinary' Residual (B)	8	8	8
Dropout Rate (p)	0.2	0.3	0.0

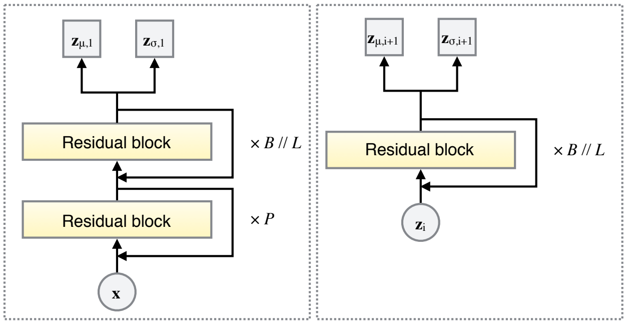


Figure 9: A schematic representation of the networks corresponding to $q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x})$ (left) and $q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$ (right) of the **inference** model. The arrows show the direction of the forward propagation.

Frey solves this problem by reversing the order of bits of the encoded $(\mathbf{z}, \mathbf{x})$ before adding it to the bitstream. This incurs a cost between 2 to 32 bits to the encoding procedure of each datapoint $\mathbf{x} \in \mathcal{D}$, depending on implementation.

D. Model Architecture

For all three datasets (MNIST, CIFAR-10 and ImageNet (32×32)), we chose a Logistic distribution $(\mu = 0, \sigma = 1)$ for the prior $p(\mathbf{z}_L)$ and conditional Logistic distributions for $q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$, $q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+l})$. The distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ is chosen to be a discretized Logistic distribution as defined in (Kingma et al., 2016). We modeled the Logistic distributions by a neural network for every pair of parameters (μ, σ). A schematic representation of the different networks is shown in Figure 9 and 10. The σ parameter of $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ is modeled unconditionally, and optimized directly. We chose Residual blocks (He et al., 2015) as hidden layers. We also used Dropout (Srivastava et al., 2014) to prevent overfitting, Weight Normalization and Data-Dependent Initialization (Salimans & Kingma, 2016), Polyak averaging

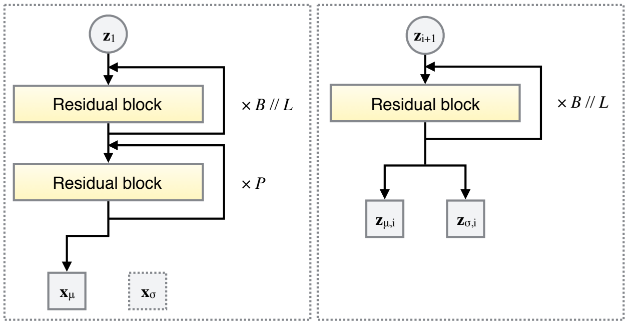


Figure 10: A schematic representation of the networks corresponding to $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ (left) and $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+l})$ (right) of the **generative** model. The arrows show the direction of the forward propagation.

(Polyak & Juditsky, 1992) of the model's parameters and the Adam optimizer (Kingma & Ba, 2015).

To make a fair comparison between the different latent layer depths L for one dataset, we used B 'ordinary' Residual blocks for the entire inference model and B for the generative model, that is kept fixed for all latent layer depths L. The blocks are distributed over the L networks that make up the inference model and L networks that make up the generative model. In addition, we added P 'processing' Residual blocks at the beginning/end of the network corresponding to $q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ respectively. Finally, we decreased the channel dimension of the output of all the Residual blocks in order to ensure that the parameter count stays constant (or regresses) as we add more latent layers. All the chosen hyperparameters are shown in Table 7. We refer to the code `https://github.com/fhkingma/bitswap` for further details on the implementation.

E. Usefulness of Latent Layers & Posterior Collapse

Posterior collapse is one of the drawbacks of using variational auto-encoders (Chen et al., 2016). Especially when using deep hierarchies of latent variables, the higher latent layers can become redundant and therefore unused (Zhao et al., 2017). We will counter this problem by using the free-bits technique as explained in (Chen et al., 2016) and (Kingma et al., 2016). As a result of this technique, all latent layers across all models and datasets are used. To demonstrate this, we generated stack plots of the number of bits/dim required per stochastic layer to encode the test set over time shown in Figure 11. The bottom-most (white) area corresponds to the bottom-most (reconstruction of $\mathbf{x}$) layer, the second area from the bottom denotes the first latent layer, the third area denotes the second latent layer, and so on.

F. Discretization of $\mathbf{z}_{1:L}$

In order to perform lossless compression with continuous latent distributions, we need to determine how to discretize the latent space $\mathbf{z}_i$ for every corresponding distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$, $q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$ and $p(\mathbf{z}_L)$. In (Townsend et al., 2019), based on (MacKay, 2003), they show that if the bins $\delta_{\mathbf{z}}$ of $\mathbf{z} \sim p(\mathbf{z})$ match the bins $\delta_{\mathbf{z}}$ of $\mathbf{z} \sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, continuous latents can be discretized up to arbitrary precision, without affecting the *net* compression rate as a result of getting ”bits back”. We generalize this result to hierarchical latent variables by stating that the bins $\delta_{\mathbf{z}_i}$ of the latent conditional generative distributions $(\mathbf{z}_1, .., \mathbf{z}_L) \sim p_{\boldsymbol{\theta}}(\mathbf{z}_1, .., \mathbf{z}_L)$ have to match the bins $\delta_{\mathbf{z}_i}$ of the inference distributions $(\mathbf{z}_1, .., \mathbf{z}_L) \sim q_{\boldsymbol{\theta}}(\mathbf{z}_1, .., \mathbf{z}_L|\mathbf{x})$ in order to avoid affecting the compression rate. Nonetheless, the length of the initial bitstream needed to decode latent sample $\mathbf{z}_1 \sim q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x})$ (or possibly samples $(\mathbf{z}_1, .., \mathbf{z}_L) \sim q_{\boldsymbol{\theta}}(\mathbf{z}_1, .., \mathbf{z}_L|\mathbf{x})$) is still dependent on the corresponding bin size(s) $\delta_{\mathbf{z}_i}$. Therefore, we cannot make the bin sizes $\delta_{\mathbf{z}_i}$ too small without affecting the total codelength too much.

There are several discetization techniques we could use. One option is to simply discretize uniformly, which means dividing the space into bins of *equal width*. However, given the constraint that the initial bitstream needed increases with larger precision, we have to make bin sizes reasonably large. Accordingly, uniform discretization of non-uniform distributions could lead to large discretization errors and this could lead to inefficient codelengths.

An option is to follow the discretization technique used in (Townsend et al., 2019) by dividing the latent space into bins that have *equal mass* under some distribution (as opposed to equal width). Ideally, the bins $\delta_{\mathbf{z}_i}$ of $\mathbf{z}_i \sim p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$ match the bins $\delta_{\mathbf{z}_i}$ of $\mathbf{z}_i \sim q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$ and the bins $\delta_{\mathbf{z}_i}$ have equal mass under either $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$ or $q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$. However, when using ANS with hierarchical latent variable models it is not possible to let the discretization of $\mathbf{z}_i \sim q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$ depend on bins based on $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$, because $\mathbf{z}_{i+1}$ is not yet available for the *sender* when decoding $\mathbf{z}_i$. Conversely, discretization of $\mathbf{z}_i \sim p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$ cannot depend on bins based on $q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$, since $\mathbf{z}_{i-1}$ is not yet available for the *receiver* decoding $\mathbf{z}_i$. Note that the operations of the compression scheme at the sender end have to be the opposite of the operations at the receiver end and we need the *same discretizations* for both ends. Under this conditions, it is not possible to use either $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$ or $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$ for the bin sizes $\delta_{\mathbf{z}_i}$ and at the same time match the bins $\delta_{\mathbf{z}_i}$ of $\mathbf{z}_i \sim p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$ with the bins $\delta_{\mathbf{z}_i}$ of $\mathbf{z}_i \sim q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$.

So, we sampled a batch from the latent generative model

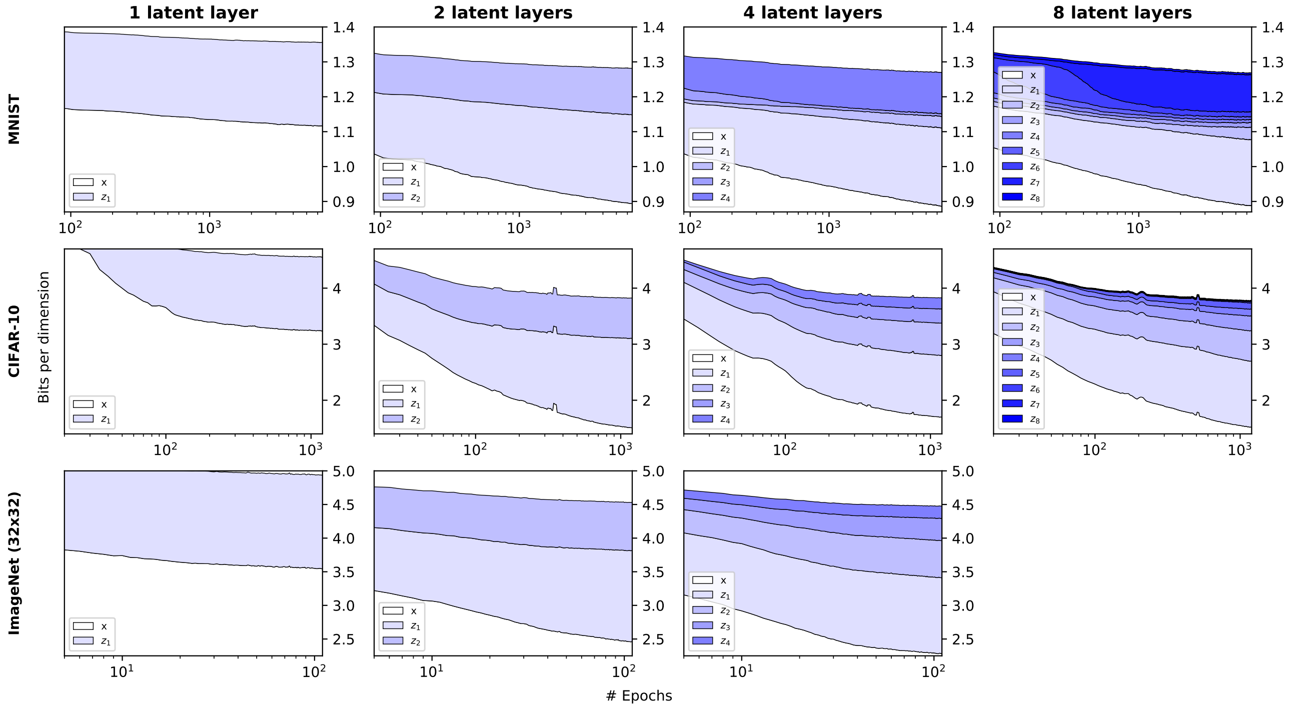


Figure 11: Stack plots of the number of bits/dim required per stochastic layer to encode the test set over time. The bottom-most (white) area corresponds to the bottom-most (reconstruction of $\mathbf{x}$) layer, the second area from the bottom denotes the first latent layer, the third area denotes the second latent layer, and so on.

$p_{\boldsymbol{\theta}}(\mathbf{z}_1, .., \mathbf{z}_L)$ by ancestral sampling and a batch from the latent inference model (using the training dataset) right after learning. This gives us unbiased estimates of the statistics of the marginal distributions $p_{\boldsymbol{\theta}}(\mathbf{z}_1), .., p(\mathbf{z}_L)$, defined in Equation 5, which we can save as part of the model. Consequently, we used the marginal distributions to determine the bin sizes for discretization of $\mathbf{z}_i \sim p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$ and $\mathbf{z}_i \sim q_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i-1})$. Note that we only have to perform this sampling operation once, hence this process does not affect the compression speed.

However, we found that using uniform discretization for all latent layers L except for the top one (corresponding to the prior) gives the best discretization and leads to the best compression results. Nevertheless, the top layer is discretized with bins that have equal mass under the prior, following Townsend et al. (2019).

G. General Applicability of Bit-Swap

Our work only concerns a very particular case of hierarchical latent variable models, namely hierarchical latent variable models in which the sampling process of both the generative- and inference model corresponding variational autoencoder obey Markov chains of the form

$$\mathbf{z}_L \rightarrow \mathbf{z}_{L-1} \rightarrow \cdots \rightarrow \mathbf{z}_1 \rightarrow \mathbf{x}$$

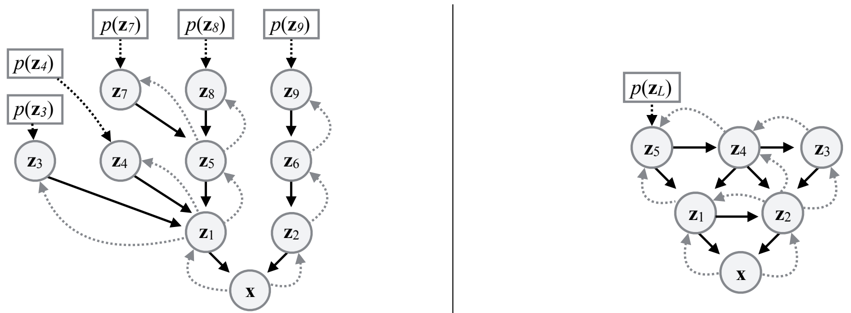
and

$$\mathbf{z}_L \leftarrow \mathbf{z}_{L-1} \leftarrow \cdots \leftarrow \mathbf{z}_1 \leftarrow \mathbf{x}$$

respectively. It might seem very restrictive to only be able to assume this topology of stochastic dependencies. However, the Bit-Swap idea can in fact be applied to any latent variable topology in which it is possible to apply the bits-back argument in a recursive manner.

To show this we present two hypothetical latent variable topologies in Figure 12. Figure 12(a) shows an asymmetrical tree structure of stochastic dependencies and Figure 12(b) shows a symmetrical tree structure of stochastic dependencies where the variables of one hierarchical layer can also be connected. In Figure 12(c) and 12(d) we show the corresponding Bit-Swap compression schemes.

The more general applicability of Bit-Swap allows us to design complex stochastic dependencies and potentially stronger models. This might be an interesting direction for future work.



(a) Asymmetrical tree structure

(b) Symmetrical tree structure including dependencies within a hierarchical layer

Sender (left) and **receiver** (right)

Sender	**Receiver**	
decode	encode	$\mathbf{z}_1$
decode	encode	$\mathbf{z}_2$
encode	decode	$\mathbf{x}$
decode	encode	$\mathbf{z}_3$
decode	encode	$\mathbf{z}_4$
decode	encode	$\mathbf{z}_5$
decode	encode	$\mathbf{z}_6$
encode	decode	$\mathbf{z}_1$
encode	decode	$\mathbf{z}_2$
encode	decode	$\mathbf{z}_3$
encode	decode	$\mathbf{z}_4$
decode	encode	$\mathbf{z}_7$
decode	encode	$\mathbf{z}_8$
decode	encode	$\mathbf{z}_9$
encode	decode	$\mathbf{z}_5$
encode	decode	$\mathbf{z}_6$
encode	decode	$\mathbf{z}_7$
encode	decode	$\mathbf{z}_8$
encode	decode	$\mathbf{z}_9$

(c) Bit-Swap executed on 12(a)

Sender (left) and **receiver** (right)

Sender	**Receiver**	
decode	encode	$\mathbf{z}_2$
decode	encode	$\mathbf{z}_1$
encode	decode	$\mathbf{x}$
decode	encode	$\mathbf{z}_3$
decode	encode	$\mathbf{z}_4$
decode	encode	$\mathbf{z}_5$
encode	decode	$\mathbf{z}_2$
encode	decode	$\mathbf{z}_1$
encode	decode	$\mathbf{z}_3$
encode	decode	$\mathbf{z}_4$
encode	decode	$\mathbf{z}_5$

(d) Bit-Swap executed on 12(b)

Figure 12: The top left Figure shows an asymmetrical tree structure of stochastic dependencies and the top right Figure shows a symmetrical tree structure of stochastic dependencies where the variables of one hierarchical layer can also be connected. The black arrows indicate the direction of the generative model and the gray dotted arrows show the direction of the inference model. The black dotted arrows show where the prior(s) is/are defined on. In the bottom left and the bottom right we show the corresponding Bit-Swap compression schemes. In the right column of every Figure, we show the variables that are being operated on. On the left of every Figure we show the operations that must be executed by the sender and in the middle we show the operations executed by the receiver. The operations must be executed in the order that is dictated by the direction of the corresponding arrow. The sender always uses the inference model for decoding and the generative model for encoding. The receiver always uses the generative model for decoding and the inference model for encoding.