Compression for audio models

Austen Lamacraft

(Dated: August 22, 2019)

CONTENTS

I. COMPRESSION

A. Compression and Entropy

The foundational result in coding theory is Shannon’s source coding theorem (the original paper Ref. [1] is well worth reading). This is a statement about the average code length for ‘messages’ described by some probability distribution p_i. The individual messages could be characters, images, audio clips: all that matters is the distribution. If we send iid messages drawn from this distribution, Shannon tells us the minimum code length per message that can be achieved *on average* i.e. in the limit of infnitely many messages. The results it that the average length in bits is bounded from below by the *entropy* $H(p)$

$$H(p) = -\sum_i p_i \log_2 p_i. \tag{1}$$

The two key questions in lossless compression are then

1. What is p_i for the data of interest?

2. How do we do the coding in practice to approach the Shannon limit as closely as possible?

Machine learning has something to offer mainly in addressing the first question. As we'll explain in a second, the bigger the chunks of data that we can model probabilistically, the better we can make our code. Consider the example that Shannon discussed in his paper: probabilistic modelling of the English language. The crudest approximation is to consider only the probability distribution $p(\text{char})$ of individual characters, pretending that each character is independent. The next approximation is to consider probabilities of neighbouring characters, described by the conditional probabilities $p(\text{char}_{t+1}|\text{char}_t)$. Adding dependence on longer and longer prefixes of characters allows us to improve the model. This shortens the code, for the following reason. If we have a model q_i for message i and we encode it with $-\log_2 q_i$ bits, the average length (averaged with respect to the *real* distribution of messages) is called the *cross-entropy*

$$H(p,q) = -\sum_i p_i \log_2 q_i = H(p) + D_{\text{KL}}\left(p\|q\right), \tag{2}$$

where

$$D_{\text{KL}}\left(p\|q\right) = \sum_i p_i \log_2\left(\frac{p_i}{q_i}\right) \geq 0 \tag{3}$$

is the Kullback–Leibler (KL) divergence. The fact that the KL divergence is positive, or equivalently that the relative entropy always exceeds the entropy is known as *Gibbs' inequality*. The implication is that coding with an imperfect distribution q means that the minimum achievable average code length increases from $H(p)$ to $H(p,q)$.

Let's return to the language example above. If we consider independent messages consisting of N-character blocks then the model $q(\text{block}) = \prod_{\text{char}\in\text{block}} p(\text{char})$ permits a minimum average code length of

$$H(p,q) = -\sum_{\substack{\text{block}\\ \text{char}\in\text{block}}} p(\text{block}) \log_2 p(\text{char}). \tag{4}$$

Note that entropies are generally proportional to the size of the message (in physics we say that entropy is an *extensive* quantity). This reflects the exponentially diminishing probabilities of any given message as the block length increases, and the associated exponential increase in number. This explosion in the number of possibilies is what limits the practicality of devising codes for ever longer blocks: you can't really consider the probability distribution over all possible images, say. However, the promise of generative models is that one can do a lot better than the simple models we discussed above and therefore get better codes adapted for a particular kind of data.

B. The 'Bits Back' Idea

The original papers on the 'bits back' argument are Refs. [2, 3] (the second one is clearer)

If you have a generative model $p(x,z) = p(x|z)p(z)$ how can you use its model likelihood

$$p(x) = \sum_z p(x|z)p(z) \tag{5}$$

for coding? The issue is that for a complicated model computing the above marginal distribution will be intractable. One possibility is to encode a pair (x_0, z_0) with z_0 coded using $p(z)$ and then x_0 encoded using $p(x|z_0)$. This involves using

$$\log p(z_0) + \log p(x_0|z_0) \tag{6}$$

bits on average. We can do better. Suppose we had access to the posterior distribution $p(z|x)$. Now, calculating $p(z|x)$ is intractable, as it involves finding $p(x)$, but we'll see in a second that having an approximate posterior is also beneficial. For now, note that since

$$\log p(x) = \log \frac{p(x|z)p(z)}{p(z|x)}, \tag{7}$$

we could begin our coding procedure by *decoding* some bits from the bitstream using $p(z|x_0)$, yielding some z_0 and reducing the message length by $-\log p(z_0|x_0)$ bits. Then we code z_0 and x_0 as before, in which case we end up with exactly $\log p(x)$ bits. Practically, this approach is suited to the situation where many z's code for a single x. Then the 'bits back' $-\log p(z|x)$ will generally be positive.

If we only have an approximation to the posterior $q(z|x)$, the same procecure yields a message of length

$$-\overbrace{\log \frac{p(x|z)p(z)}{q(z|x)}}^{\text{ELBO}} \geq -\log p(x), \tag{8}$$

with equality only achieved if $q(z|x) = p(z|x)$. The negative of the quantity on the left is called the *Evidence Lower Bound* (or ELBO) and is the central quantity in variational inference.

C. Normalizing Flows

The central idea of normalizing flows is to assume that the data of interest x are sampled from a highly complex probability distribution $x \sim p_X$, and to learn a transformation that maps this distribution to a simpler one. If we map to new variables z by a function $z = f(x)$ then the p_X is mapped transformed according to

$$p_Z(z) = p_X(x) \left| \frac{dx}{dz} \right|, \tag{9}$$

where $\left|\frac{dx}{dz}\right|$ is the determinant of the Jacobian matrix $J = \frac{dx}{dz}$ describing the transformation. If we fix a distribution $p_Z(z)$ then the mapping f defines a model distribution $p_X(x)$. Usually we take $p_Z(z)$ to be factorized into identical distributions

$$p_Z(z) = \prod_j \pi(z_j), \tag{10}$$

so that the components z_j are iid random variables (standard Gaussians say $z_j \sim \mathcal{N}(0,1)$).

The model is specified by the parameters that described f, usually expressed in terms of neural networks, and traditionally the function corresponds to the composition of several simpler functions $f = f_N \circ f_{N-1} \circ \cdots f_1$, each with a tractable Jacobian J_k $k = 1, \ldots N$, such that the overall Jacobian determinant is $|J| = \prod_k |J_k|$. If we don't take trouble to choose functions with 'nice' Jacobians, the computation of a general determinant is $O(N^3)$, where N is the dimensionality of the data.

Training a normalizing flow (or any generative model) amounts to maximizing the log likelihood of the model averaged over the data

$$\frac{1}{N_{\text{data}}} \sum_{i \in \text{data}} \log p_X(x^{(i)}) = \mathop{\mathbb{E}}_{x \sim p_{\text{data}}} [\log p_X(x)] = -H(\text{data}, \text{model}), \tag{11}$$

which we recognize as minus the cross entropy. This is therefore maximized when the model and data distributions coincide.

A normalizing flow thus provides a model of p_X. How would this allow us to compress the data?

D. Flows and Compression

1. *Integer (and Discrete) Flows*

Most normalizing flow models developed so far have operated with continuous variables. If we want to use a trained flow model for compression, we need to work with discrete variables. Integer [4] and discrete [5] flow models have recently been developed. The Integer Discrete Flows (IDF) paper [4] specifically deals with the subject of compression. The *key difference* from the continuous case is the absence of a Jacobian, so that

$$p_Z(z = f(x)) = p_X(x). \tag{12}$$

The cross entropy (11) is then

$$H(\text{data}, \text{model}) = \frac{1}{N_{\text{data}}} \sum_{i \in \text{data}} \log p_Z(f(x_i)) = \frac{1}{N_{\text{data}}} \sum_{i \in \text{data}} \sum_{k=1}^{N} \log \pi(f_k(x^{(i)})) \tag{13}$$

for N-dimensional data, assuming $p_Z(z) = \prod_k \pi(z_k)$. *If* you manage to map the data distribution to the factored distribution exactly, this will achieve the minimum $H(\text{data})$ (obviously this statement only applies to the $N_{\text{data}} \to \infty$ limit). In fact the IDF paper chooses a more flexible model by conditioning the latents factored out at each stage of the flow on the variables not factored out $\pi(z_j|\theta_j(y))$. θ_j denote the parameters of the distribution of the j^{th} component: in the paper they use a mixture of discretized logistic distributions. According to Rianne van den Berg, the reasons for this choice *vs.* the option of learning a general discrete distribution via softmax are

1. Softmax imposes no ordering whereas we general want an ordered set (e.g. pixel values).

2. Softmax would have many parameters, compared with just the mean and standard deviation of the logistic (repeated K times for a mixture).

3. Not so easy to handle the absence of an upper interval.

4. Sparse gradients.

[For reference PixelCNN did softmax; PixelCNN++ did mixture of logistics. I think the same was true with WaveNet vs. Parallel WaveNet]

2. Bits back

An alternative, presented in Ref. [6], is to train a *continuous* flow model, where the Jacobian determinant relates the probability densities. The assumption (*I believe*) is that $|J| \ll 1$, indicating that the latent densities are smaller as the data is 'spread out' into the latent space. With the same discretization, there will then be places where many points in the latent space are mapped to a single point in the data space. Simply coding in the latent space using the probabilities $p_Z(\bar{z})\delta_z$ – where $\bar{z}$ denotes the quantized values and δ_z is the volume of a cell in the latent space – ignores this redundancy. The procedure described in [6] essentially estimates this redundant volume and recovers the bits describing it via the bits back procedure.

One objection to the approach of Ref. [6] may be that the overall map is not going to be well approximated by a linear map, justifying the Gaussian treatment. However, they apply the bits back scheme at each level of the mapping and this may be a way around.

I have the feeling that the schemes of Refs. [4, 6] are not so different. Learning the hierarchical distribution is probably performing the same function as the bits back scheme.

3. Bits back as variational inference

How does the procedure of [6] fit into the general scheme described in Section I B? When normalizing flows are defined for *continuous* variables the conditional distributions $p(x|z)$ and $p(z|x)$ are deterministic, being described by $x = f^{-1}(z)$ and $z = f(x)$ respectively. However, when we naively discretize such a mapping, there will be places where the bijection breaks down as different inputs are mapped to the same output. In this way we end up with two mappings $\bar{z} = \lfloor f(\bar{x}) \rceil$ and $\bar{x} = \lfloor f^{-1}(\bar{z}) \rceil$, where we follow Ref. [6] in denoting the integer part by a bar. In general these maps are not bijections, and one is not the inverse of the other. Both are surjective.

If we regard the decoder map $\bar{x} = \lfloor f^{-1}(\bar{z}) \rceil$ as $p(\bar{x}|\bar{z})$ then the corresponding $p(\bar{z}|\bar{x})$ is probabilistic. $p(\bar{z}|\bar{x})$ is intractable for high dimensional data: we can't afford to search over $\bar{z}$ to find all values that map to $\bar{x}$. To apply the bits back scheme we therefore need an approximate posterior $q(\bar{z}|\bar{x})$, which is the role played by the discretized Gaussian in [6].

I think the assumption of this approach to compression is that it is $\bar{x} = \lfloor f^{-1}(\bar{z}) \rceil$ that tends to have regions of many-to-one behaviour, while $\bar{z} = \lfloor f(\bar{x}) \rceil$ is predominately injective. If this wasn't the case, so that there were many places where $\bar{z} = \lfloor f(\bar{x}) \rceil$ is many-to-one, the bits back scheme won't lead to effective compression.

4. Bits back in general

Suppose we have a model consisting of a pair of conditionals $q(z|x)$ and $q(x|z)$ (transition kernels) between integer valued (or generally discrete) data, with tractable distributions that encode with and decode (sample) from, together with a latent distribution $p(z)$. The bits back coding scheme is then

- Start from symbol x_0.
- Decode z_0 from random bits using $q(z|x_0)$.
- Encode x_0 using $q(x|z_0)$.
- Encode z_0 using $p(z)$

The overall number of bits is

$$\mathop{\mathbb{E}}_{\substack{x_0\sim\text{data}\\ z_0\sim q(z|x_0)}} \log \frac{q(z_0|x_0)}{q(x_0|z_0)p(z_0)}. \tag{14}$$

This is minimized when $q(z|x)$ and $q(x|z)$ are transition kernels between $p(z)$ and $p_{\text{data}}(x)$

$$p(z) = \sum_x q(z|x)p_{\text{data}}(x) \tag{15}$$

$$p_{\text{data}}(x) = \sum_x q(x|z)p(z). \tag{16}$$

To see this, rewrite (14) as

$$\mathop{\mathbb{E}}_{\substack{x_0\sim\text{data}\\ z_0\sim q(z|x_0)}} \log \frac{q(z_0|x_0)}{q(x_0|z_0)p(z_0)} = \mathop{\mathbb{E}}_{\substack{x_0\sim\text{data}\\ z_0\sim q(z|x_0)}} \log \frac{q(z_0|x_0)p_{\text{data}}(x)}{q(x_0|z_0)p(z_0)} - \mathop{\mathbb{E}}_{x_0\sim\text{data}} \log p_{\text{data}}(x_0) \tag{17}$$

$$= D_{\text{KL}}\left(q(z|x)p_{\text{data}}(x) \| q(x|z)p(z)\right) + H(\text{data}). \tag{18}$$

The first term is the KL divergence between the two joint distributions $q(z|x)p_{\text{data}}(x)$ and $q(x|z)p(z)$. It is positive, and vanishes when the two distributions coincide, so that (15) is satisfied. At this point we achieve the Shannon minimum.

Notes

- This objective is really just the usual VAE objective, it's just normally presented as minimizing over a variational $q(z|x)$ given some generative model $p(x|z)$. VAEs can suffer from 'posterior collapse', meaning that the $q(z|x)$ ends up spreading over the latent space if the model is too expressive, so that $q(x|z)$ approach p_{data}. This is bad if you want interpretable latent dimensions, but there's nothing wrong with it from the coding point of view: it means you get more bits back! In fact, the expressiveness of the models will be limited by the need to encode and decode with $q(z|x)$ and especially $q(x|z)$. If you were able to code with a model $q(x|z)$ that approached $p_{\text{data}}(x)$ there would be no need for a generative model in the first place!

- When the transition kernels describe a bijection

$$q(z|x) = \delta(z - f(x)) \tag{19}$$

$$q(x|z) = \delta(x - f^{-1}(z)) \tag{20}$$

  we have

$$\frac{q(z|x)}{q(x|z)} = \left|\frac{\partial x}{\partial z}\right|, \tag{21}$$

  and we recover the objective for a normalizing flow. Although Ref. [6] add noise for the purpose of encoding / decoding, they still train with the original objective. See Ref. [7] for more on the connection between VAEs and NFs.

- We do still need to be able to take gradients through the expectations over z. This suggests a continuous z is still preferable, even for discrete x. As shown in Ref. [8] the resolution in the z space cancels out as it appears in both $q(z|x)$ and $p(z)$. Alternatively, one can use the score function estimator, see Section II A.

- Jittering, often done to dequantize discrete data, effectively adds noise to a normalizing flow. This is essentially what is done in Ref. [6].

- It is possible in principle to use separate networks for $q(z|x)$ and $q(x|z)$ (c.f. parallel WaveNet). However, since encoding and decoding require both, it's necessary in practice that both are fast.

5. *From lossless to lossy compression*

Following the IDF scheme, one possible way to interpolate from lossless to lossy compression is to keep only higher level latents and sample the lower ones. Because their distribution depends on the higher level variables this will preserve the learned distribution.

E. Applications of NNs to Audio Compression

[9, 10] explore lossy compression of speech with neural networks.

Ref. [11] on the IEEE AAC standard says that state of the art lossless compression is only $2\times$. But this is *hopeless*! Imagine what a signal with that entropy sounds like! It would be almost indistinguishable from white noise. Lossy compression that exploits perceptual shrortcomings of human hearing gets $20\times$ compression.

II. WAVENET AUTOENCODER

With a view to bits back encoding for audio models we consider a probabilistic encoder $p_\theta(z|x)$ and decoder $p_\phi(x|z)$ for integer valued data with causal structure

$$p_\theta(z|x) = \prod_t p_\theta(z_t|x_{1:t-1}) \tag{22}$$

$$p_\phi(x|z) = \prod_t p_\phi(x_t|z_{1:t-1}). \tag{23}$$

The distributions $p_\theta(z_t|x_{1:t-1})$ and $p_\phi(x_t|z_{1:t-1})$ could be mixtures of discretized logistics, say, as for PixelCNN++ and Parallel WaveNet [12]. The decoder is an example of *inverse autoregressive flow* and the encoder can be thought of as the inverse pass of a *masked autoregressive flow* (see Jang's blog).

The VAE objective to be minimized takes the form

$$\mathcal{L}_{\text{VAE}} = \mathop{\mathbb{E}}_{\substack{x\sim\text{data}\\ z\sim p_\theta(\cdot|x)}} \log \frac{p_\theta(z|x)}{p_\phi(x|z)p(z)} = \mathop{\mathbb{E}}_{\substack{x\sim\text{data}\\ z\sim p_\theta(\cdot|x)}} \left[-\log p(z) + \sum_t \left(\log p_\theta(z_t|x_{1:t-1}) - \log p_\phi(x_t|z_{1:t-1})\right)\right]. \tag{24}$$

A. Gradients

How do we optimize with respect to parameters θ and ϕ? Gradients with respect to ϕ are straightforward, as they just appear inside the expectation. Gradients with respect to θ are trickier, as θ appears in the distribution used for the expectation. When VAEs use continuous latent variables, this issue is usually handled using the *reparameterization trick.* For example, if $\boldsymbol{z}$ is Gaussian $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ then z may be written in terms of a vector of iid standard normals $\xi_j \sim \mathcal{N}(0,1)$ as

$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{\xi} + \boldsymbol{\mu} \tag{25}$$

where $\boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}^T$. Thus we would parameterize $\boldsymbol{A}_\theta$ and $\boldsymbol{\mu}_\theta$, average with respect to $\boldsymbol{\xi}$, and substitute (25) inside the expectation. Differentiating with respect to θ is then as easy as for ϕ.

Unfortunately, for discrete variables this isn't an option: one can't continuously change a sample from a discrete distribution. One alternative is to use the REINFORCE algorithm or score function estimator (see this blog post). Despite the long names the idea is simple and is based on the identity

$$\frac{\partial}{\partial\theta} \mathop{\mathbb{E}}_{z\sim p_\theta} [f(z)] = \mathop{\mathbb{E}}_{z\sim p_\theta} \left[f(z) \frac{\partial \log p_\theta(z)}{\partial\theta} \right]. \tag{26}$$

In this way we get

$$\frac{\partial}{\partial\theta} \mathop{\mathbb{E}}_{\substack{x\sim\text{data}\\ z\sim p_\theta(\cdot|x)}} [\log p_\phi(x_t|z_{1:t-1})] = \mathop{\mathbb{E}}_{\substack{x\sim\text{data}\\ z\sim p_\theta(\cdot|x)}} \left[\left(\sum_{t_1} \log p_\phi(x_{t_1}|z_{1:t_1-1}) \right) \frac{\partial}{\partial\theta} \left(\sum_{t_2} \log p_\theta(z_{t_2}|x_{1:t_2-1}) \right) \right]. \tag{27}$$

PyTorch implementation here. Although the score function estimator is simple there can be issues with the variance. See Ref. [13] for a recent review about Monte Carlo gradient estimation.

The VAE objective (24) also involves the entropy of the encoder distribution

$$\mathop{\mathbb{E}}_{z\sim p_\theta(\cdot|x)} \log p_\theta(z|x) = -H(q_\theta(\cdot|x)). \tag{28}$$

It's possible that this can be computed analytically in terms of the the parameters of the distribution. For a categorical distribution you just have to sum over the values. I don't know how to do this for the mixture of discretized logistics, but perhaps it would suffice to use a lower bound on entropy for a mixture $q(z) = \sum_j c_j q_j(z)$

$$H(q) \geq \sum_j c_j H(q_j). \tag{29}$$

See TensorFlow docs for discussion.

B. Concrete Implementation

I suggest implementing the encoder p_θ and decoder p_ϕ as WaveNets. Usually, WaveNet is implemented as a pure autoregressive model $p(x_t|x_{1:t-1})$ with no latent variables, but I think there is nothing to stop us switching z_t for x_t.

The original WaveNet [14] used 8 bit audio (256 levels) and a categorical distribution via SoftMax. Parallel WaveNet [12] used 16 bit audio and a mixture of logistic distributions.

NVIDIA has a reference implementation of WaveNet with a PyTorch wrapper. Whether the CUDA implementation is necessary with GPU support in PyTorch I don't know. Details are described in this blog post. This is the autoregressive model only, without conditioning, so is suitable for us.

I would start with the 8 bit case and a categorical distribution.

C. Contrast with parallel WaveNet

Parallel WaveNet [12] (see also Jang's blog) also used two networks, but the strategy is different. They first train a network ('teacher') as before but then use another network ('student') for generation, training the student's distribution to match the teachers.

D. Bits back with autoregressive encoder / decoder

[1] C. E. Shannon, Bell system technical journal **27**, 379 (1948).

[2] G. E. Hinton and R. S. Zemel, in *Advances in neural information processing systems* (1994) pp. 3–10.

[3] B. J. Frey and G. E. Hinton, in *Proceedings of Data Compression Conference-DCC'96* (IEEE, 1996) pp. 73–81.

[4] E. Hoogeboom, J. W. Peters, R. v. d. Berg, and M. Welling, arXiv preprint arXiv:1905.07376 (2019).

[5] D. Tran, K. Vafa, K. K. Agrawal, L. Dinh, and B. Poole, arXiv preprint arXiv:1905.10347 (2019).

[6] J. Ho, E. Lohn, and P. Abbeel, arXiv preprint arXiv:1905.08500 (2019).

[7] A. A. Gritsenko, J. Snoek, and T. Salimans, (2019).

[8] J. Townsend, T. Bird, and D. Barber, arXiv preprint arXiv:1901.04866 (2019).

[9] S. Kankanahalli, in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2018) pp. 2521–2525.

[10] C. Gârbacea, A. van den Oord, Y. Li, F. S. Lim, A. Luebs, O. Vinyals, and T. C. Walters, in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2019) pp. 735–739.

[11] H. Huang, H. Shu, and R. Yu, in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2014) pp. 6934–6938.

[12] A. v. d. Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. v. d. Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, *et al.*, arXiv preprint arXiv:1711.10433 (2017).

[13] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, arXiv preprint arXiv:1906.10652 (2019).

[14] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, arXiv preprint arXiv:1609.03499 (2016).