# Learning Wavelets For Compression

Austen Lamacraft

(Dated: August 9, 2019)

# Abstract

Many compression strategies start with a transformation to decorrelate the signal and render it more suitable for entropy encoding. Normalizing flows – a class of generative model that learns an invertible transformation from the data distribution to a factorized distribution – are therefore a natural candidate for transform coding that can be adapted to the data. Wavelet transforms are often used in this context. Here we investigate the possibility of learning a wavelet transform in the form of a lifting scheme adapted to integer valued data.


**CONTENTS**



## I. BACKGROUND

### 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. 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

$$p_Z(z) = p_X(x) \left| \frac{dx}{dz} \right|, \tag{5}$$

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

$$p_Z(z) = \prod_j \pi(z_j), \tag{6}$$

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

$$\frac{1}{N_{\text{data}}} \sum_{i \in \text{data}} \log p_X(x_i) = \mathop{\mathbb{E}}_{x \sim p_{\text{data}}} [\log p_X(x)] = -H(\text{data}, \text{model}), \tag{7}$$

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?

### C. 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 [2] and discrete [3] flow models have recently been developed. The Integer Discrete Flows (IDF) paper [2] specifically deals with the subject of compression.

A flow maps a complicated data distribution to a factorized distribution. A factorized distribution allows each component of the representation $z = f(x)$ to compressed separately according to its distribution $\pi(z_j)$. However, if we fix the distribution $\pi(z)$ at the outset, we can never do better than the entropy of this factorized distribution, which is

$$H_Z = -N \sum_z \pi(z) \log \pi(z) \tag{8}$$

for $N$-dimensional data. We would never gain anything by learning the mapping to $z$! To deal with this, the IDF paper conditions the latents factored out at each stage of the flow on the variables not factored out $\pi(z_j|\theta_j(y))$. $\theta_j$ denote the parameters of the distribution of the $j^{\text{th}}$ 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. [4], 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 $\delta_z$ is the volume of a cell in the latent space – ignores this redundancy. The procedure described in [4] essentially estimates this redundant volume and recovers the bits describing it via the bits back procedure.

I believe that to get effective compression one has to adapt the latent distribution or use bits back.

*3. 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.

### D. Applications of NNs to Audio Compression

[5, 6] explore lossy compression of speech with neural networks.

Ref. [7] on the IEEE AAC standard says that state of the art lossless compression is only 2×. 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× compression.

---


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

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

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

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

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

[6] 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.

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