Lossless Source Coding

Gadiel Seroussi  Marcelo Weinberger

Information Theory Research
Hewlett-Packard Laboratories
Palo Alto, California
Notation

- $A = \text{discrete (usually finite) alphabet}$
- $\alpha = |A| = \text{size of } A$ (when finite)
- $x_1^n = x^n = x_1 x_2 x_3 \cdots x_n = \text{finite sequence over } A$
- $x_1^\infty = x^\infty = x_1 x_2 x_3 \cdots = \text{infinite sequence over } A$
- $x_i^j = x_i x_{i+1} \cdots x_j = \text{sub-sequence} (i \text{ sometimes omitted if } i = 1)$
- $p_X(x) = \text{Prob}(X=x) = \text{probability mass function (PMF) on } A$
  (subscript $X$ and argument $x$ dropped if clear from context)
- $X \sim p(x): X \text{ obeys PMF } p(x)$
- $E_p[F] = \text{expectation of } F \text{ w.r.t. PMF } p$ (subscript and $[\ ]$ may be dropped)
- $\hat{p}_{x_1^n}(x) = \text{empirical distribution obtained from } x_1^n$
- $\log x = \text{logarithm to base 2 of } x$, unless base otherwise specified
- $\ln x = \text{natural logarithm of } x$
- $H(X), H(p) = \text{entropy of a random variable } X \text{ or PMF } p$, in bits; also
  
  $H(p) = -p \log p - (1-p) \log (1-p), \ 0 \leq p \leq 1$ : binary entropy function
- $D(p||q) = \text{relative entropy (information divergence) between PMFs } p \text{ and } q$
Lossless Source Coding

3. Universal Coding
Universal Modeling and Coding

So far, it was assumed that a model of the data is available, and we aimed at compressing the data optimally w.r.t. the model.

By Kraft’s inequality, the models we use can be expressed in terms of probability distributions:
For every UD code with length function $L(s)$, we have

$$\sum_{s \in A^n} 2^{-L(s)} \leq 1$$

a code defines a probability distribution $P(s) = 2^{-L(s)}$ over $A^n$.

Conversely, given a distribution $P(\cdot)$ (a model), there exists a UD code that assigns $\lceil -\log p(s) \rceil$ bits to $s$ (Shannon code).

Hence, $P(\cdot)$ serves as a model to encode $s$, and every code has an associated model.

A (probabilistic) model is a tool to “understand” and predict the behavior of the data.
Universal Modeling and Coding (cont.)

q Given a model $P(\cdot)$ on $n$-tuples, arithmetic coding provides an effective mean to **sequentially** assign a code word of length close to $-\log P(s)$ to $s$

- we don’t need to see the whole string $s = x_1 \ldots x_n$: encode $x_t$ using conditional probability

$$p(x_t \mid x_1 x_2 \ldots x_{t-1}) = \frac{P(x^t)}{P(x^{t-1})} = \frac{\sum_{u \in A^{n-t}} P(x^t u)}{\sum_{v \in A^{n-t+1}} P(x^{t-1} v)}$$

- the model probabilities can vary arbitrarily and “adapt” to the data
- in a **strongly** sequential model, $P(x^t)$ is independent of $n$

CODING SYSTEM = MODEL + CODING UNIT

- two separate problems: design a model and use it to encode

We will view data compression as a problem of assigning probabilities to data
Coding with Model Classes

q **Universal data compression** deals with the optimal description of data in the absence of a given model
   - in most practical applications, the model is not given to us
q How do we make the concept of “optimality” meaningful?
   - there is always a code that assigns just 1 bit to the data at hand!
   **The answer: Model classes**
q We want a “universal” code to perform as well as the best model in a **given** class \( C \) for any string \( s \), where the best competing model changes from string to string
   - universality makes sense only w.r.t. a model class
q A code with length function \( L(x^n) \) is **pointwise universal** w.r.t. a class \( C \) if
   \[
   R_C(L, x^n) = \frac{1}{n} [L(x^n) - \min_{C \in C} L_C(x^n)] \to 0 \quad \text{when} \quad n \to \infty
   \]
   pointwise redundancy
How to Choose a Model Class?

Universal coding tells us how to encode optimally w.r.t. to a class; it doesn’t tell us how to choose a class!

Some possible criteria:
- complexity
- prior knowledge on the data
- some popular models were already presented

We will see that the bigger the class, the slower the best possible convergence rate of the redundancy to 0
- in this sense, prior knowledge is of paramount importance: don’t learn what you already know!

Ultimately, the choice of model class is an art
Example: Bernoulli Models

\[ A = \{0,1\}, \quad C = \{P_\theta, \theta \in \Lambda = [0,1]\} \]

\[ \min_{C \in C} \log \frac{1}{P_\theta(x^n)} = \log \frac{1}{P_{\theta(x^n)}(x^n)} = nH(x^n) \]

Therefore, our goal is to find a code such that

\[ \frac{L(x^n)}{n} - \frac{1}{H(x^n)} \rightarrow 0 \]

Here is a trivial example of a universal code: Use \([\log(n+1)]\) bits to encode \(n_1\), and then “tune” your Shannon code to the parameter \(\theta = n_1/n\), which is precisely the ML-estimate of \(\theta\)!
Bernoulli Models: Enumerative Code

q The total code length for this code satisfies:

\[
\frac{L(x^n)}{n} \leq \left( \frac{n_0}{n} \right) \log \left( \frac{n_0}{n} \right) - \left( \frac{n_1}{n} \right) \log \left( \frac{n_1}{n} \right) + \frac{\log(n+1) + 2}{n} = \hat{H}(x^n) + \frac{\log(n+1) + 2}{n} \rightarrow 0
\]

\[\text{wasteful: knowledge of } n_1 \text{ already discards part of the sequences, to which we should not reserve a code word}\]

q A slightly better code: enumerate all \( \binom{n}{n_1} \) sequences with \( n_1 \) ones, and describe the sequence with an index

\[
\frac{L'(x^n)}{n} = \frac{1}{n} \left[ \log \binom{n}{n_1} \right] \leq \hat{H}(x^n) + \frac{\log(n+1) + 2}{n}
\]

Stirling: \( \binom{n}{n_1} \leq \frac{2^{n\hat{H}(x^n)}}{\sqrt{2\pi (n_1 n_0) / n}} \Rightarrow \text{for sequences such that } n_1/n \text{ is bounded away from 0 and 1, } \frac{L'(x^n)}{n} \leq \hat{H}(x^n) + \frac{\log n}{2n} + O(1/n)\]
Bernoulli Models: Mixture Code

q The enumerative code length is close to \(-\log Q'(x^n)\), where

\[ Q'(x^n) = \binom{n}{n_1}^{-1} \cdot \frac{1}{n+1} \]

q Is \(Q(\cdot)\) a probability assignment?

\[ \int_0^1 P_\theta(x^n) d\theta = \int_0^1 \theta^{n_1} (1-\theta)^{n_0} d\theta = \binom{n}{n_1}^{-1} \cdot \frac{1}{n+1} = Q'(x^n) \Rightarrow \sum_{x^n \in A^n} Q'(x^n) = 1 \]

q Aha!! So we get the same result by mixing all the models in the class and using the mixture as a model!

q This uniform mixture is very appealing because it can be sequentially implemented, independent of \(n\):

\[ Q'(x^n) = \frac{n_0! n_1!}{(n+1)!} = \prod_{t=0}^{n-1} q'(x_{t+1} | x^t) \text{ where } q'(0 | x^t) = \frac{n_0(x^t) + 1}{t + 2} \]

- Laplace’s rule of succession!
- this results in a “plug-in” approach: estimate \(\theta\) and plug it in
Bernoulli Models: Mixture Code (cont.)

q Maybe we can make Stirling work for us with a different mixture, that puts more weight in the problematic regions where $n_1/n$ approaches 0 and 1?

q Consider Dirichlet’s density

$$w(\theta) = \frac{1}{\Gamma(\frac{1}{2})^2 \sqrt{\theta (1-\theta)}}$$

$$Q''(x^n) = \int_0^1 P_\theta (x^n) dw(\theta) = \frac{1}{\Gamma(\frac{1}{2})^2} \int_0^1 \theta^{n_1 - \frac{1}{2}} (1-\theta)^{n_0 - \frac{1}{2}} d\theta = \frac{\Gamma(n_0 + \frac{1}{2}) \Gamma(n_1 + \frac{1}{2})}{n! \Gamma(\frac{1}{2})^2}$$

⇒ by Stirling for the Gamma function, for all sequences $x^n$

$$\frac{L''(x^n)}{n} = -\frac{1}{n} \log Q''(x^n) \leq \hat{H}(x^n) + \frac{\log n}{2n} + O(1/n)$$

Can we do any better?

q This mixture also has a plug-in interpretation:

$$Q''(x^n) = \prod_{i=0}^{n-1} q''(x_{i+1} | x^t)$$

where

$$q''(0 | x^t) = \frac{n_0(x^t) + \frac{1}{2}}{t + 1}$$
Summary of Codes

q Two-part code
   - describe best parameter and then code based on it: works whenever
     the number of possible optimal codes in the class is \textit{sub-exponential}
   - the most natural approach (akin to Kolmogorov complexity), but not
     sequential
   - not efficient in the example: can be improved by describing the
     parameter more coarsely \Rightarrow trade-off: spend less bits on the parameter
     and use an \textit{approximation} of best parameter
   - alternatively, don’t allocate code words to impossible sequences

q Mixture code
   - can be implemented sequentially
   - a suitable \textit{prior} on the parameter gave the smallest redundancy

q Plug-in codes
   - use a biased estimate of the conditional probability in the model class
     based on the data seen so far
   - can often be interpreted as mixtures

q But what’s the best we can do?
Normalized Maximum-Likelihood Code

q Goal: find a code that attains the best worst-case pointwise redundancy

\[ R_C = \min_L \max_{x^n \in A^n} R_C(L, x^n) = \frac{1}{n} \min_L \max_{x^n \in A^n} [L(x^n) - \min_{C \in C} L_C(x^n)] \]

q Consider the code defined by the probability assignment

\[ Q(x^n) = \frac{2^{-\min_{C \in C} L_C(x^n)}}{\sum_{x^n \in A^n} 2^{-\min_{C \in C} L_C(x^n)}} \Rightarrow R_C(L, x^n) = \frac{1}{n} \log \left( \sum_{x^n \in A^n} 2^{-\min_{C \in C} L_C(x^n)} \right) \]

This quantity depends only on the class!

q Since the redundancy of this code is the same for all \( x^n \), it must be optimal in the minimax sense!

q **Drawbacks**: - sequential probability assignment depends on horizon \( n \) - hard to compute
Example: NML Code for the Bernoulli Class

We evaluate the minimax redundancy $R_C$ for the Bernoulli class:

$$R_C = \frac{1}{n} \log \left[ \sum_{x^n \in A^n} 2^{-H(x^n)} \right] = \frac{1}{n} \log \left[ \sum_{i=0}^{n} \binom{n}{i} 2^{-h(i/n)} \right] = \frac{1}{2n} \log \frac{n\pi}{2} + o(1/n)$$

typical of sufficient statistics:

$$P_\theta (x^n) = p(x^n | s(x^n)) p_\theta (s(x^n))$$

Hint: Stirling +

$$\int_0^1 \frac{d\theta}{\sqrt{\theta (1-\theta)}} = \Gamma \left( \frac{1}{2} \right)^2 = \pi$$

⇒ the pointwise redundancy cannot vanish faster than $(\log n)/2n$

(Dirichlet mixture or NML code)
Parametric Model Classes

q A useful limitation to the model class is to assume \( C = \{ P_{\theta}, \theta \in \Theta_d \} \)

\[ a \text{ parameter space of dimension } d \]

q Examples:

- Bernoulli: \( d = 1 \), general i.i.d. model: \( d = \alpha - 1 \) (\( \alpha = |A| \))
- FSM model with \( k \) states: \( d = k(\alpha - 1) \)
- Memoryless geometric distribution on the integers \( i \geq 0 \): \( P(i) = \theta^i (1-\theta) \)

q The dimension of the parameter space (number of one-dimensional parameters) plays a fundamental role in modeling problems

- with more parameters we can better fit the model to the data (e.g., a Markov model of higher order \( \Rightarrow \) the entropy cannot increase)
- but on the other hand, the class is richer and the redundancy is higher: e.g., in a two-part code it takes more bits to describe the best parameter

we will quantify this tradeoff of the model selection step
Minimax Redundancy for Parametric Classes

For a parametric class, $\min_{C \in C} L_C(x^n) = -\log P_{(x^n)}(x^n)$

ML-estimate of $\theta$ based on $x^n$

A code corresponding to a distribution $Q(x^n)$, has a redundancy

$$\log \frac{P_{(x^n)}(x^n)}{Q(x^n)}$$

and the minimax redundancy (with the NML code) is

$$R_C = \frac{1}{n} \log \left[ \sum_{x^n \in A^n} P_{(x^n)}(x^n) \right]$$

the more sensitive the class to the parameter, the larger the redundancy
Assumptions on the Parametric Model Classes

q We will assume that each model in the class satisfies the marginality condition for a random process, namely
\[ \sum_{x_{n+1} \in A} P_\theta (x^{n+1}) = P_\theta (x^n) \]
this means that each model can be implemented in a strongly sequential manner

q We will usually assume that the model class is “nice”, including:

\( \Theta_d \) is an open bounded subset of \( \mathbb{R}^d \) which includes the ML estimates

the Fisher information matrix \( I(\theta) \) is “nice” \( I_{ij}(\theta) = -\lim_{n \to \infty} \frac{1}{n} E \left[ \frac{\partial^2 \ln P_\theta (x^n)}{\partial \theta_i \partial \theta_j} \right] \)
for i.i.d., this is the classical Fisher information

the maximum-likelihood estimator satisfies the CLT: the distribution of \( \sqrt{n} (\hat{\theta} (x^n) - \theta) \) converges to \( N(0, I^{-1}(\theta)) \)
Redundancy of NML Code

q Theorem: 
\[ R_C = \frac{d}{2n} \log \frac{n}{2\pi} + \frac{1}{n} \log \int_{\Theta_d} \sqrt{I(\theta)} \, d\theta + o(1/n) \]

As expected, \( R_C \) grows with the number of parameters

q Idea of the proof:
- partition the parameter space into small hypercubes with sides \( r = o(1/\sqrt{n}) \)
- represent each hypercube \( V_r(\theta^r) \) by a parameter \( \theta^r \) belonging to it
- associate to \( V_r(\theta^r) \) the mass 
  \[ P_r(\theta^r) = \sum_{\theta'(x^n) \in V_r(\theta^r)} P_{\theta^r}(x^n) \]
- with a Taylor expansion of \( P_\theta(x^n) \) (as a function of \( \theta \)) around its maximum \( \theta'(x^n) \) we get the exact asymptotics of \( R_C \rightarrow \sum_{\theta^r} P_r(\theta^r) \)
- for each hypercube, approximate \( P_r(\theta^r) \) using CLT \( \Rightarrow \) integral of a normal distribution
Interpretation of NML as a Two-part Code

q Encode $x^n$ in two parts:

1. **Part I**: encode the parameter $\theta^r$ for the hypercube $V_r(\theta^r)$ into which $\hat{\theta}(x^n)$ falls, with a code tuned to the distribution $\frac{P_r(\theta^r)}{\sum_{\theta^r} P_r(\theta^r)}$

2. **Part II**: encode $x^n$ using the model $\hat{\theta}^r$, conditioned on the fact that the ML estimate belongs to $V_r(\hat{\theta}^r)$ \Rightarrow distribution $\frac{P_{\theta^r}(x^n)}{P_r(\theta^r)}$

q In earlier two-part codes, the parameters were represented with a fixed precision $O(1/\sqrt{n})$ and then $x^n$ was encoded based on the approximate ML parameter

1. evaluating the approximation cost with Taylor, this precision was shown to be optimal

1. intuitively, $1/\sqrt{n}$ is the magnitude of the estimation error in $\hat{\theta}(x^n)$ and therefore there is no need to encode the estimator with better precision

$$\frac{d}{2} \log n = \text{MODEL COST}$$
Redundancy of the Mixture code

\[
Q_w(x^n) = \int_{\theta \in \Theta_d} P_{\theta}(x^n) dw(\theta)
\]

$q$ As in the Bernoulli example, for any i.i.d. exponential family, and for Markov models, mixture code also gives the same “magic” model cost when the distribution \( w(\theta) \) is proportional to \( \sqrt{|I(\theta)|} \) (Jeffrey’s prior)

- in the Bernoulli case, this is precisely Dirichlet’s distribution: \( I(\theta) = \frac{1}{\theta (1-\theta)} \)
- the tool for solving the integral is Laplace’s integration method

$q$ The advantage of mixtures is that they yield sequential horizon-free codes:

\[
\sum_{x_t \in A} Q_w(x^t) = \int_{\theta \in \Theta_d} \sum_{x_t \in A} P_{\theta}(x^t) dw(\theta) = \int_{\theta \in \Theta_d} P_{\theta}(x^{t-1}) dw(\theta) = Q_w(x^{t-1})
\]

\[
q_w(x_t | x^{t-1}) = \frac{Q_w(x^t)}{Q_w(x^{t-1})} = \int_{\theta \in \Theta_d} P_{\theta}(x_t | x^{t-1}) \frac{P_{\theta}(x^{t-1})}{\int_{\theta \in \Theta_d} P_{\theta}(x^{t-1}) dw(\theta)} dw(\theta)
\]
Important Example: FSM model classes

Given FSM $S$ with $k$ states (fixed initial state)
parameters = conditional probabilities per state $p(x|s)$, $x \in A$, $s \in S$

ML estimates:

$$p_{x^n}(x | s) = \frac{n_{x^n}(x|s)}{n_{x^n}(s)}$$

target: $\min_{\theta \in \Theta_{k(\alpha-1)}} \log \frac{1}{P_{\theta}(x^n)} = \log \frac{1}{P_{\theta}(x^n)(x^n)} = nH(x^n | S)$

mixture distribution:

$$Q_w(x^n) = \int_{\theta \in \Theta_{k(\alpha-1)}} \prod_{s \in S, x \in A} p(x|s)^{n_{x^n}(x|s)} \, dw(\theta) = \prod_{s \in S, x \in A} \int_0^1 \theta^{n_{x^n}(x|s)} \, dw(\theta)$$

$\Rightarrow$ equivalent to doing i.i.d. mixture for every state sub-sequence

$\Rightarrow$ optimal universal probability assignment:

$$q_{x^n}(x | s) = \frac{n_{x^n}(x|s) + 1/2}{n_{x^n}(s) + \alpha/2}$$

Krichevski-Trofimov (KT) estimator
FSM model: Numerical Example

$q$ FSM = First order Markov, initial state 0

$x^{20} = 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0$

\begin{align*}
&1/2 \ 3/4 \ 1/6 \ 1/2 \ 3/4 \ 5/6 \ 1/8 \ 3/8 \ 7/10 \ 3/4 \ 11/14 \ 13/16 \ 5/6 \ 3/20 \ 1/2 \ 7/12 \ 9/14 \ 5/16 \ 5/22 \ 13/18
\end{align*}

Total probability:

\begin{align*}
Q(x^n) &= \prod_{s \in S} \frac{\Gamma(n(0|s) + \frac{1}{2})\Gamma(n(1|s) + \frac{1}{2})}{n(s)!\Gamma(\frac{1}{2})^2} = \frac{\Gamma(6.5)\Gamma(3.5)\Gamma(3.5)\Gamma(8.5)}{9!1!\Gamma(\frac{1}{2})^4} \\
- \log Q(x^{20}) &\approx 17.61 \quad \Rightarrow \quad \text{code length: 18 bits}
\end{align*}

\begin{align*}
\mathcal{H}(x^{20} \mid S) = 9h(1/3) + 11h(3/11) &\approx 17.56 \quad \text{bits}
\end{align*}
The LZ algorithm is universal for ANY class of FSM models (of any size) ⇒ in particular, for all Markov models

\[
\limsup_{n \to \infty} \frac{1}{n} L_{LZ}(x^n) \leq \limsup_{n \to \infty} H(x^n | S_k)
\]

and for an infinite sequence \(x^\infty\)

\[
\limsup_{n \to \infty} \frac{1}{n} L_{LZ}(x^n) \leq \lim_{k \to \infty} \lim_{n \to \infty} H(x^n | S_k) = H(x^\infty)
\]
Expected Pointwise Redundancy

q So far, we have considered the code length for individual sequences, without taking any average.

q We have found that for parametric classes, the best we can do is to achieve a worst-case pointwise redundancy of about \((d/2)\log n\) bits.

q This means guaranteed performance, but maybe there are only a few such “unlucky” strings?

q How to weight the redundancies in order to compute an average?

Since we are assuming that the class \(C = \{P_\theta, \theta \in \Theta_d\}\) is a good model for the data, it makes sense to assume that the data was drawn from a source with some distribution \(P_\theta\) in \(C\).

q Expected pointwise redundancy of a code

\[
E_\theta \left[ R_C(L, x^n) \right] = \frac{1}{n} E_\theta \left[ L(x^n) + \log P_\theta(x^n) (x^n) \right]
\]
Expected Redundancy

Maybe there exist codes that are “good” (with expected pointwise redundancy smaller than \((d/2)\log n\)) for a significant fraction of models in the class?

We will be even “larger”: we consider the expected redundancy

\[
\overline{R}_C(L, \theta) = \frac{1}{n} \mathbb{E}_\theta \left[ L(x^n) \right] - H_n(\theta) = D_n(P_\theta \parallel Q)
\]

normalized divergence for distr. defined on \(n\)-tuples

This is more “liberal” because

\[
H_n(\theta) = \frac{1}{n} \mathbb{E}_\theta \left[ -\log P_\theta(x^n) \right] \geq \frac{1}{n} \mathbb{E}_\theta \left[ -\log P_{\theta(x^n)}(x^n) \right]
\]

and therefore

\[
\overline{R}_C(L, \theta) \leq \mathbb{E}_\theta \left[ R_C(L, x^n) \right]
\]
The pointwise universal codes we saw are - a fortiori - average universal for all parameters \( \theta \Rightarrow \) this means that the corresponding distribution \( Q \) is “close” to all the models in the class in the sense that \( D_n(P_\theta \parallel Q) \rightarrow 0 \)

Still, the answer is that \((d/2)\log n\) is about the best we can do for most models in the class: it’s the inevitable cost of universality

Theorem:
Assume that either CLT holds for ML estimator of parameters in \( \Theta_d \) or \( \Pr\{\sqrt{n}(\theta_i(x^n) - \theta_i) \geq \log n\} \leq \delta(n) \rightarrow 0 \)
Then for all \( Q \) and all \( \varepsilon > 0 \),
\[
- n^{-1} E_\theta [\log Q(x^n)] \geq H_n(\theta) + k \frac{\log n}{2n} (1 - \varepsilon)
\]
for all points \( \theta \) in \( \Theta_d \) except in a set whose volume \( \rightarrow 0 \) as \( n \rightarrow \infty \)
Lower Bound (cont.)

q This lower bound parallels Shannon’s coding theorem: when we consider a model class instead of a single distribution, a model cost gets added to the entropy.

q The bound cannot hold for all models in the family, but it holds for most.

q One interpretation of the lower bound: if the parameters can be estimated well, they are “distinguishable” ($P_\theta$ is sensitive to $\theta$), so the class cannot be coded without a model cost.

**Conclusion:**

the number of parameters affects the achievable convergence rate of a universal code length to the entropy.
Variations on the Lower Bound

q Assuming, in addition, that $\sum_n \delta(n) < \infty$, then the cumulative volume of “bad” parameters for all sufficiently large $n \to 0 \Rightarrow$ the “bad” parameters form a set of Lebesgue volume 0

q **Strong Redundancy-Capacity Theorem:**
Under very mild conditions on the model class,

$$- n^{-1} E_\theta [\log Q(x^n)] \geq H_n(\theta) + (1 - \varepsilon)C_n$$

where $C_n = \sup_w \left[ H_n(Q_w) - \int_{\Theta} H_n(\theta)d\omega(\theta) \right] \geq 0$ and $Q_w$ is a mixture,

for all $\theta$ except for a set $B$ for which $w^*(B) \leq e^{2^{-nC_n\varepsilon}}$, where $w^*$ is the density that achieves the supremum

q For parametric classes, $C_n$ indeed behaves as $(d/2n)\log n$ and $w^*(B) \to 0$
Optimal Codes for Average Redundancy

q How do we achieve \( \inf_L \sup_\theta \left\{ E_\theta \left[ L(x^n) \right] - H_n(\theta) \right\} \)?

We use the mixture \( Q_w \) for which the \textit{weighted} expected redundancy
\[
\int_{\theta \in \Theta_d} \left[ E_\theta \left[ -\log Q_w(x^n) \right] - H_n(\theta) \right] dw(\theta)
\]
is maximum \( \Rightarrow C_n \)

\( \Rightarrow \) an appropriate mixture is the best one can do to minimize the
expected redundancy for the worst-case parameter (close to
Jeffrey’s prior in the case of exponential families/Markov models)

q In many situations, the minimum expected redundancy (or at least
its main term \( (d/2) \log n \)) can be achieved by a \textit{plug-in} code of the
form
\[
Q(x^n) = \prod_{t=1}^n P_{\tilde{\theta}_{t-1}}(x_t)
\]
where \( \tilde{\theta}_{t-1} \) is an estimate of \( \theta \) based on \( x^{t-1} \)

- example: Bernoulli/FSM cases (KT estimator; even Laplace’s estimator
  works in the average sense)
Summary of Codes and Bounds

q Minimax pointwise redundancy: NML code

for “nice” parametric families with \( d \) parameters,

\[
R_C = \frac{d}{2n} \log \frac{n}{2\pi} + \frac{1}{n} \log \int_{\Theta_d} \sqrt{|I(\theta)|} d\theta + o(1/n)
\]

horizon-dependent, mixture code with suitable prior is a good horizon-free approximation

in fact, for i.i.d. models with \( \alpha \)-1 parameters, it can be shown that

\[
\lim_{n \to \infty} q_{NML}(x_t \mid x_1 x_2 \ldots x_{t-1}) = \lim_{n \to \infty} \frac{1}{n} \sum_{u \in A^{n-t}} Q_{NML}(x' u) = \frac{n x_t(x \mid s) + 1/2}{n x_t(s) + \alpha/2}
\]

KT mixture!

\[
\begin{align*}
\text{KT mixture!}
\end{align*}
\]
Summary (cont.)

q Minimax average redundancy: mixture codes with prior such that the weighted expected redundancy is maximum

\[ C_n = \sup_w \left[ H_n(Q_w) - \int_{\Theta} H_n(\theta) d\theta \right] \]

for i.i.d. exponential families and Markov models, close to Jeffrey’s prior

\[ C_n = \frac{d}{2n} \log \frac{n}{2\pi e} + \frac{1}{n} \log \int_{\Theta_d} \sqrt{I(\theta)} d\theta + o(1/n) \]

not only we cannot do better than \( C_n \) for one parameter (minimax), but for most parameters

for “nice” parametric families,

\[ C_n \approx \frac{1}{2} \frac{d \log n}{n} \]
Twice-universal coding

q Consider a model class to be the union of nested classes of growing dimensionality: \( \Theta_1 \cup \Theta_2 \cup \ldots \cup \Theta_d \cup \ldots \)
where \( \Theta_1 \subset \Theta_2 \subset \ldots \subset \Theta_d \subset \ldots \)
- example: Markov of different order, or FSM

q We seek a code length close to the minimum over the classes of the universal code length for each class:

\[
\frac{1}{n} L(x^n) \approx \min_d \min_{\theta \in \Theta_d} \left\{ H_n(\theta) + \frac{d}{2n} \log n \right\}
\]

- here, we also try to optimize the model size
- trade-off: first part diminishes with \( d \), second part grows with \( d \)
- we answer questions such as: should we model the data as i.i.d. or as Markov of order 1?
- since the second level of universality is over a discrete set, we expect it not to affect the main redundancy term
Mixture Approach to Double-Universality

q Consider the series $1 = \sum_i \lambda_i$ where $\lambda_i > 0$

The probability assignment $Q(x^n) = \sum_d \lambda_d Q_d(x^n)$

is sequential: $Q(x^n) = \sum_{x_{n+1}} Q(x^{n+1})$

It is universal as desired: for all $d$

$$-\frac{1}{n} \log Q(x^n) \leq -\frac{1}{n} \log Q_d(x^n) - \frac{\log \lambda_d}{n} < \min_{\theta \in \Theta_d} \log \frac{1}{P_\theta(x^n)} + \frac{d}{2n} \log n + O(1/n)$$

q However, it involves an infinite sum

- this can be avoided by modifying $Q_d$ so that $Q_d(x_t) = \alpha^{-1}$ for all $t \leq d$:
  - this way, all models with $d \geq n$ assign the same probability $\alpha^n$ and the sum becomes finite, without affecting universality and sequentiality

- anyway, this is obviously not practical but shows achievability of the goal
Plug-in Approach to Double-Universality

1. The idea: estimate the best model class $\Theta_d$ based on $x^{t-1}$, and plug it to encode $x_t$ with $Q_d$

$$Q(x^n) = \prod_{t=1}^{n} Q_{d(x^{t-1})}(x_t | x^{t-1})$$

- the decoder can reproduce the process without overhead

2. In general, this approach does not work pointwise, but for some model classes it was shown to work on the average
Lossless Source Coding

5. Universal Algorithms for Tree Models
Double-Universality for Tree Models

**Tree sources (FSMX)**
- finite memory $\leq k$ (Markov)
- # of past symbols needed to determine the state might be $< k$ for some states

![Tree diagram](image)

- by merging nodes from the full Markov tree, we get a model with a smaller number of free parameters
- the set of tree sources with unbalanced trees has measure zero in the space of Markov sources of any given order: otherwise, double-universality would contradict the lower bound!!
- yet, tree models have proven very useful in practice, partly because there exist efficient twice-universal, sequential schemes in the class of tree models of any size
Any suffix of a sequence $x^t$ is called a context in which the next symbol $x_{t+1}$ occurs.

For a finite-memory process $P$, the conditioning states $s(x^t)$ are contexts that satisfy $P(a \mid x^t) = P(a \mid us(x^t))$, $\forall u \in A^*$, $a \in A$.

# of parameters: $\alpha-1$ per leaf of the tree

- assume minimal parametrization: sibling states with identical conditional distribution are “merged”
Two-pass Context Algorithm

- **First pass:** gather all the context statistics for \( x^n \) in a tree
  - \( O(n) \) complexity with suffix tree methods
- **After the first pass,** associate to each node a cost
  
  \[
  L(s) + \frac{\alpha}{\alpha - 1}
  \]

  KT code length for symbols occurring at \( s \): includes penalty for overparametrization

  and “prune” the tree to find a subtree \( T \) with total minimum cost for the leaves (dynamic programming algorithm)

- **The total cost for** \( T \) **is:**
  
  \[
  L_T(x^n) + \frac{1}{\alpha - 1} = L_T(x^n) + n_T + \frac{1}{\alpha - 1}
  \]
Two-pass Context Algorithm (cont.)

q Second pass: describe $T$ to the decoder using $n_T$ bits and encode $x^n$ conditioned on $T$ with $KT \Rightarrow$ by definition, we have the best tree

$$\frac{1}{n} L(x^n) = \frac{1}{n} L_T(x^n) + \frac{n_T}{n} < H(x^n \mid T) + \frac{(\alpha - 1)T}{2n} \log n + O(1/n)$$
Pruning of Context Trees: Example

Tree used for encoding: the best model to the data seen so far. Could change dynamically with each input symbol.

counts

\( n(a|s) \)

\( \hat{p}(a|s) \)

typical node

Data structure maintained by the algorithm: a superset of the tree used for encoding.
Drawbacks of Two-pass Approach

- Non-sequential

- The scheme has asymptotically minimum pointwise redundancy, but is redundant (given the tree, not all $x^n$ are possible)

$$Q(x^n) = 2^{-L(x^n)} = 2^{-L_T(x^n)} 2^{-n_T} = 2^{-L_T(x^n)} \lambda_T < \sum_T \lambda_T 2^{-L_T(x^n)}$$

⇒ mixture with weights $\lambda_T$ is better!

- Can the mixture be efficiently done?
Mixture Approach: Context Tree Weighting

- **CTW**: An efficient implementation of the mixture for the class of binary tree models of maximum length bounded by \( m \).
  - There exist extensions for non-binary and unbounded, but not as elegant.

- **Probability assignment associated with a node** \( s \) **of the tree**

\[
Q^w_T(s) = \begin{cases} 
  Q^w_T(s) + Q^w_T(s0)Q^w_T(s1) \\
  Q^w_T(s) \text{ if depth}(s) = m
\end{cases}
\]

\[
q_{CTW}(x_t | x_1x_2...x_{t-1}) = \frac{Q^w_T(x^t)}{Q^w_{CTW}(x^{t-1})} = \frac{Q^w_T(\lambda)}{Q^w_{T-1}(\lambda)}
\]

- **Theorem**: Let \( T \) have \( l_T \) leaves (\( \Rightarrow n_T = 2^{l_{T-1}} \), \( m_T \) leaves at level \( m \).

\[
Q_{CTW}(x^t) = \sum_T 2^{-2l_T + 1 + m_T} Q^T_K(x^t) \quad \Rightarrow \quad L_{CTW}(x^n) \leq L^T_K(x^t) + n_T
\]

KT prob. assignment conditioned on \( T \) for any \( T \)
Let $m = 2$

$$Q^w(\lambda) = \frac{Q^{KT}(\lambda) + Q^w(0)Q^w(1)}{2}$$

$$= \frac{Q^{KT}(\lambda)}{2} + \frac{[Q^{KT}(0) + Q^w(00)Q^w(01)][Q^{KT}(1) + Q^w(10)Q^w(11)]}{8}$$

$$= \frac{Q^{KT}(\lambda)}{2} + \frac{Q^{KT}(0)Q^{KT}(1)}{8} + \frac{Q^{KT}(0)Q^{KT}(10)Q^{KT}(11)}{8}$$

root $\implies n_T - m_T = 3$

$$= \frac{Q^{KT}(1)Q^{KT}(01)Q^{KT}(00)}{8} + \frac{Q^{KT}(00)Q^{KT}(01)Q^{KT}(10)Q^{KT}(11)}{8}$$

$\implies n_T - m_T = 3$

$$Q^w_t(s) = \begin{cases} 
Q^{KT}_t(s) + Q^w_t(s0)Q^w_t(s1) \\
2Q^{KT}_t(s) & \text{if depth}(s) = m
\end{cases}$$
CTW: Practical Issues

q Finite precision: as node probabilities get smaller they cannot be stored ⇒ may need to re-compute from counts
- probability assignment is a ratio of vanishing numbers

q In general, $m$ needs to be assumed large, and complexity raises

q Binary only

q However, there are good implementations of CTW which provide the best compression ratios on text and binary file compression

q From a practical viewpoint, a plug-in approach is more appealing
  however, universality is only shown on an average sense
Plug-in Approach

In a plug-in approach, the idea is to encode $x_{t+1}$ by selecting a context from $x^t$ that strikes the “right balance” between conditional entropy and model cost:

- A long context reduces entropy by introducing more conditioning.
- But the longer the context, the smaller occurrence counts it has and the statistics that we learn from them are less reliable.
- For each time $t$, it is only necessary to select a context in the path $x_t, x_{t-1}, x_{t-2}, \ldots$, not the whole tree.

A plug-in twice-universal scheme consists of a context selection rule, and a coding scheme based on the statistics stored at the selected context (e.g., KT estimator).

The context selection rule is also a tool for model selection for purposes other than data compression.
**Context Algorithm**

- The algorithm consists of three interleaved stages:
  - growing of a tree that captures, essentially, all occurrences of each symbol at each node
  - a context selection rule that selects for $x_{t+1}$ a context $s_t(x^t)$ from the tree $T_t$ grown by $x^t$
  - a KT sequential probability assignment for $x_{t+1}$ based on the counts stored at $s_t(x^t)$

- For each new symbol, we first encode, then update $T_t \rightarrow T_{t+1}$ (think of the decoder!)
  - the update consists of incrementing the occurrence counts for all the nodes in the path $x_t x_{t-1} x_{t-2} \ldots$
  - when we get to a leaf that was already visited, we extend the tree one level and initialize the count of $x_{t+1}$ to 1 (the others remain 0) ⇒ only a few initial occurrences are missing, so we can basically assume that for all nodes $s$ and all symbols $a$, the count $n_{x^t}(a|s)$ is available
  - nodes in other paths don’t need to be visited
Context Selection Rule

- Basic principle: 
  *The node which would have assigned the shortest code length for its symbol occurrences in the past string should be selected*

- Most intuitive choice: find minimum cost tree $T_t$ for all times $t$, and do KT-coding for $x_{t+1}$ conditioned on the context $x_t x_{t-1} x_{t-2} \ldots$ in $T_t$
  - Unlike two-pass, the cost should *not* include tree description: the decoder already has it!
  - Very complex and was not shown to be asymptotically optimal, even on the average

- Another possibility: for each node $sb \in T_t$, $b \in A$ define
  \[
  \Delta_t(sb) = L_t(sb) - L_t'(s)
  \]
  KT code length for symbols occurring at $sb$ based on counts gathered at $s$
  KT code length for symbols occurring at $sb$ 

  Choose the deepest node $sb$ in $T_t$ such that $\Delta_t(sb) < 0$
Universality of Context Algorithm

q Easier to analyze:

\[ \Delta_t(sb) = \sum_{a \in A} n_{x_t}(a \mid sb) \log \frac{p_t(a \mid sb)}{p_{x_t}(a \mid s)} \]

the selected tree \( T_t \) is the smallest complete super-tree of

\{ the deepest nodes \( w \) in \( T_t \) s.t. \( \Delta_t(w) \geq C \log (t+1) \} \)

q Theorem: For any minimal complete tree \( T \) with \( k \) leaves defining a tree source \( P_T(x^n) \) with probabilities bounded away from 0, if \( C > 2(\alpha+1) \) then the probability assignment \( Q \) of Context Algorithm satisfies

\[ \frac{1}{n} E_T[\log \frac{P_T(x^n)}{Q(x^n)}] \leq k(\alpha - 1) \frac{\log n}{2n} + O(1/n) \]

and, moreover,

\[ \frac{1}{n} \log \frac{P_T(x^n)}{Q(x^n)} \leq k(\alpha - 1) \frac{\log n}{2n} + O(1/n) \]

with \( P_T \)-probability 1
Idea of the Proof

q The proof is based on the fact that

\[ \sum_{i=1}^{\infty} P_T \{ x^i \mid T^i \neq T \} \log t < \infty \]

and so the contribution of the “bad” sequences is \( O(1/n) \)

q Two (non-disjoint) classes of errors:

- **overestimation**: the selected tree contains an internal node which is a leaf of the true tree; taken care of by the penalty term

- **underestimation**: a leaf of the selected tree is an internal node of the true tree; requires large deviation techniques
PPM algorithm

- A context selection rule very popular in the data compression community: choose the shortest context that occurred more than a certain number of times
  - rationale: the context gathered enough statistics in order to be reliable
  - the rule is totally *ad hoc*: if the best tree model is short, it will tend to overestimate (think of data generated by an i.i.d. source!)

- A family of algorithms based on variations of this selection rule is called PPM (*Prediction by Partial Matching*)
  - it is a very popular scheme for text compression and yields some of the best compression ratios
  - however, algorithms with a stronger theoretical basis tend to do better
Application to Statistics: the MDL Principle

- Model selection is probably the most important problem in statistical inference.

- Minimum Description Length (MDL) principle of statistical inference: 
  
  *choose the model class that provides the shortest code length for the model and the data in terms of the model* ⇒ *universal coding theory provides the yardstick to measure this code length*

- Rationale: models serve as tools to describe regularities in the data.
  - We should use the simplest explanation for the data, but not too simple.

- Strong consistency shown in various settings, solves problem of model order selection avoiding the use of artificial penalty terms.

- Bayesian interpretation through mixture codes.
  - However, NML code cannot be explained as a mixture.

- Other interpretations: maximum-entropy principle.
Lossless Source Coding

6. Sequential Decision Problems
The Sequential Decision Problem

q The framework
   l observations: $x^n = x_1 x_2 \ldots x_n, x_t \in A$
   l corresponding actions: $b^n = b_1 b_2 \ldots b_n, b_t \in B$
   l instantaneous losses $l(b_t, x_t)$ accumulate over time:
     \[ L(x^n) = \sum_{t=1}^{n} l(b_t, x_t) \]

q On-line (sequential) strategy
   l \{b_t\}, action $b_t$ is decided before observing $x_t$
   l possibly randomized: \{ $p_t(b_t | x^{t-1}, b^{t-1})$ \}

q The goal: as $n \to \infty$, approximate performance of best strategy in a given reference class, for arbitrary $x^n$ (deterministic setting)
   l excess loss w.r.t. reference: regret or redundancy
   l the reference class (or expert set) may reflect limited resources
Prediction with expert advice

Most general setting: reference class = set of generic “experts”

Basic principle for on-line strategy

select an expert’s prediction randomly, with probability dependent on its accumulated loss

Goal:
\[ L_{\text{seq}}(x^n) \leq L_{\text{min}}(x^n) + \varepsilon(n) \]
Sequential Decision Problem: Examples

q Binary prediction with Hamming loss

- $x_1 x_2 \ldots x_n$ is a binary sequence ($|A|=2$)
- the action: predict either $b_t = 0$ or $1$ (deterministic), or assign a probability $p_t$ to $1$ (randomized strategy) ($|B|=2$)
- the loss:
  - deterministic strategy: $l(b_t, x_t) = 0$ if $b_t = x_t$ and $1$ otherwise
  - accumulated loss = total # of prediction errors
  - randomized strategy: $E[ l(b_t, x_t)] = |x_t - p_t|$

q A less trivial example: lossless data compression

- $x_1 x_2 \ldots x_n$ is the data to encode, finite alphabet $A$
- the (deterministic) action $b_t$ is a probability distribution assigned to $x_t$
  - $b_t = \{p_t(\cdot | x_1 x_2 \ldots x_{t-1})\}$ ($B$ continuous and vectorial!)
- the loss: $l(b_t, x_t) = -\log p_t(x_t | x_1 x_2 \ldots x_{t-1})$ (given the assigned distribution, an encoder can generate a code word of length $l(b_t, x_t)$)
- the accumulated loss (total code length) is $-\log$ of the probability assigned to $x_1 x_2 \ldots x_n$
Exponential-Weighting Algorithm

q The most general scheme for on-line expert selection
   here, we will assume $A$, $B$ finite and loss bounded by $l_{\text{max}}$

q $L_F(x^t) = \text{loss of expert } F \in F \text{ accumulated through time } t$

At time $t+1$ choose the action suggested by expert $F$ with probability

$$P_{t+1}(F) = \frac{e^{-\eta L_F(x^t)}}{\sum_{F' \in F} e^{-\eta L_{F'}(x^t)}}$$

some given positive constant

number of experts

Then,

$$\mathbb{E}_{\text{ew}}(x^n) \leq \min_{F \in F} L_F(x^n) + \frac{\ln \beta}{\eta} + \frac{n \eta l_{\text{max}}^2}{8}$$

$\Rightarrow$ with $\eta = \frac{1}{l_{\text{max}}} \sqrt{(8 \ln \beta)/n}$, normalized excess loss over best expert is

$$l_{\text{max}} \sqrt{(\ln \beta)/(2n)} \quad \text{(horizon-dependent scheme)}$$
Horizon-Free Exponential Weighting

q Horizon-free variant: divide the data into blocks of exponentially growing size, and apply the horizon-dependent algorithm to each block $\Rightarrow$ it is easy to see that the overall loss increases only by a constant factor for all $n$
Example: Binary Prediction with Constant Experts

- Two experts: one always says “0”, the other always says “1”
  - analogous to memoryless model in data compression
- \( L_{\text{min}}(x^n) = \min(n_0, n_1) \)
  \[
  P_{t+1}(0) = \frac{1}{1 + e^{-\eta [n_0(x') - n_1(x')]}},
  \]

Horizon-free approximation:

Choose the current winner except if up to \( \varepsilon(t) \) from tie, with \( \varepsilon(t) = O(\sqrt{1/n}) \), in which case randomize ⇒ similar redundancy, different constant

- Minimum worst-case redundancy: draw \( x_{t+1} x_{t+2} \ldots x_n \) at random and predict the winner in the overall sequence of length \( n-1 \)
FS reference strategies (L-Z framework)

- A given FSM is driven by the observations \( \{x_t\} \)
  
  \[ S = \text{set of states} \quad f = \text{next-state function} \]
  
  \[ s_1 = \text{fixed initial state} \quad s_{t+1} = f(s_t, x_t) \]

  Reference strategy is allowed to vary following the FSM: \( b_t = g(s_t) \)

  - example: one-state machine = constant strategy

- Best \( g \) for the specific \( x^n \):
  
  \[ g(s) = \arg \min_{b \in B} \mathbb{E}_x \left[ l(b, x) \mid s \right] \]

  expectation w.r.t. conditional empirical distribution

  \( \Rightarrow \) normalized regret vanishes by applying single-state strategy to sub-sequences at each state

- Take best FSM for \( x^n \), consider normalized loss for \( n \to \infty \), and \( |S| \to \infty \)

- For log loss: FS “compressibility” of an infinite individual sequence
  
  - efficiently achievable: LZ data compression scheme
Similarly, FS “predictability”
sequential “LZ-like” decision scheme performs essentially as well as the best FSM (of any size!) for the sequence

Example: $x^{12} = 0 1 01 00 010 011$

“context” at which 8th decision is made

$n_0$

$n_1$
**Average Loss**

- When the goal is to minimize the *average* number of prediction errors, the redundancy can be made much smaller than the worst pointwise case.
  - This is different from data compression (log loss)!

- For example, for binary prediction and two constant experts, “0” and “1”, the redundancy is given by $n^{-1}E_\theta[\#\text{errors}]\cdot\min(\theta,1-\theta)$ and it is $O(1/n)$ for a majority predictor *without* randomization.

**Proof:** Assuming (without loss of generality) $p(1) = \theta < 0.5$,

$$P_t(\text{error}) = \theta P_\theta \{ \hat{x}_t = 0 \} + (1-\theta)P_\theta \{ \hat{x}_t = 1 \} = \theta + (1-2\theta)P_\theta \{ \hat{x}_t = 1 \} \Rightarrow$$

$$E_\theta[\#\text{errors}] = \sum_{i=1}^{n} P_t(\text{error}) = n\theta + (1-2\theta)\sum_{i=1}^{n} P_\theta \{ \hat{x}_t = 1 \} \Rightarrow$$

$$\frac{1}{n} E_\theta[\#\text{errors}] - \theta < \frac{(1-2\theta)}{n} \sum_{i=1}^{\infty} P_\theta \{ n_{\hat{x}}(1) \geq n_{\hat{x}}(0) \} = O(1/n)$$

vanishes exponentially fast (Chernoff)
Lossless Source Coding

7. Lossless image compression
Lossless Image Compression (the real thing…)

Some applications of lossless image compression:

- Images meant for further analysis and processing (as opposed to just human perception)
  - Medical, space
- Images where loss might have legal implications
  - Medical
- Images obtained at great cost
- Applications with intensive editing and repeated compression/decompression cycles
- Applications where desired quality of rendered image is unknown at time of acquisition

A new international standard (ISO/IEC: “JPEG Committee”): JPEG-LS
Universality vs. Prior Knowledge

- Application of universal algorithms for tree models directly to real images yields poor results
  - some structural symmetries typical of images are not captured by the model
  - a universal model has an associated “learning cost:” why learn something we already know?

- Modeling approach: limit model class by use of “prior knowledge”
  - for example, images tend to be a combination of smooth regions and edges
  - predictive coding was successfully used for years: it encodes the difference between a pixel and a predicted value of it
  - prediction errors tend to follow a Laplacian distribution \( \Rightarrow \) AR model + Laplacian, where both the center and the decay are context dependent
  - Prediction = fixed prediction + adaptive correction
Models for Images

- Continuous tone images
  - Gray scale: a 2D array of pixel intensity values (integers) in a given range \([0..(\alpha -1)]\) (often \(\alpha=256\))
  - Color: a 2D array of vectors (e.g. triplets) whose coordinates represent intensity in a given color space (e.g., RGB, YUV); similar principles

- In practice, contexts are formed out of a finite subset of the past sequence

- Conditional probability model for prediction errors: two-sided geometric distribution (TSGD)

\[
P(e) = c_0 \theta^{|e+s|}, \quad \theta \in (0,1), \quad s \in [0,1)
\]

- “discrete Laplacian”
- shift \(s\) constrained to \([0,1)\) by integer-valued adaptive correction (bias cancellation) on the fixed predictor
Complexity Constraints

- Are sophisticated models worth the price in complexity?
  - Algorithm Context and CTW are linear time algorithms for tree sources of limited depth, but quite expensive in practice
  - Even arithmetic coding is not something that a practitioner will easily buy in many applications!
- Is high complexity required to approach the best possible compression?
- The idea in JPEG-LS: apply judicious modeling to reduce complexity, rather than to improve compression

the modeling/coding separation paradigm is less neat without complex models or arithmetic coding
The LOCO-I algorithm

- JPEG-LS is based on the LOCO-I algorithm:  
  **LOw COmplexity LOssless COmpression of Images**

- **Basic components:**
  - Fixed + Adaptive prediction
  - Conditioning contexts based on quantized gradients
  - Two-parameter conditional probability model (TSGD)
  - Low complexity adaptive coding matched to the model (variants of Golomb codes)
  - **Run length coding** in flat areas to address drawback of symbol-by-symbol coding
JPEG-LS (LOCO-I Algorithm): Block Diagram

- **Gradients**
- **Flat Region?**
- **Predictor**
- **Context Modeler**
- **Golomb Coder**
- **Run Counter**
- **Run Coder**

**Image samples** → **mode** → **regular** → **Pred. values** → **Pred. errors** → **Golomb Coder**

**Image samples** → **Pred. errors** → **Context Modeler**

**Pred. values** → **Predictor**

**Pred. errors, code spec.** → **Golomb Coder**

**Run lengths, code spec.** → **Run Coder**

**Regular** → **compressed bit-stream**
Fixed Predictor

- **Causal template** for prediction and conditioning

  \[
  x_{i+1} = \begin{cases} 
  \min(a, b) & \text{if } c \geq \max(a, b) \\
  \max(a, b) & \text{if } c \leq \min(a, b) \\
  a + b - c & \text{otherwise}
  \end{cases}
  \]

  median of \(a, b, \) and \(a+b-c\)

- **Nonlinear**, has some “edge detection” capability:
  - Predicts \(b\) in “vertical edge”
  - Predicts \(a\) in “horizontal edge”
  - Predicts \(a+b-c\) in “smooth region”
Parameter Reduction and Adaptivity

- The goal in selecting the number of parameters: capture high order dependencies without excessive model cost
- Adaptive coding is needed, but arithmetic coding ruled out (if possible…) due to complexity constraints
- A solution that addresses both issues: model prediction residuals with a TSGD
  - only two parameters per context
    - “sharpness” (rate of decay, variance, etc.)
    - shift (often non-zero in a context-dependent scheme)
  - allows for large number of contexts (365 in JPEG-LS)
  - suited to low complexity adaptive coding

\[
P(e) = c_0 \theta^{|e+s|}, \quad \theta \in (0,1), \quad s \in [0,1)
\]
Context Determination

Causal template:

\[
\begin{array}{ccc}
  c & b & d \\
  a & & x \quad \text{next pixel}
\end{array}
\]

Look at the gradients \( g_1 = (d-b), \ g_2 = (b-c), \ g_3 = (c-a), \)
- gradients capture the level of activity (smoothness, edginess) surrounding a pixel
- \( g_1, g_2, g_3 \) quantized into 9 regions determined by 3 thresholds \( S_1, S_2, S_3 \)
- maximum information on \( x_{i+1} \) suggests equiprobable regions

<table>
<thead>
<tr>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>-S_3</td>
<td>-S_2</td>
<td>-S_1</td>
<td>0</td>
<td>S_1</td>
<td>S_2</td>
<td>S_3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symmetric contexts merged:

\[
P(\varepsilon | [q_1, q_2, q_3]) \leftrightarrow P(-\varepsilon | [-q_1, -q_2, -q_3])
\]

A fixed number of contexts: \( (9^3 + 1)/2 = 365 \)
Optimal prefix codes for TSGDs are built out of the Golomb codes for nonnegative integers.

Given a code parameter $m$ and an integer $j$,

$$j \rightarrow \{ j \mod m \text{ (in binary), } \lfloor j/m \rfloor \text{ (in unary) } \}$$

Example: $j = 19, m = 4$:

\[
\begin{array}{c|c}
19 \mod 4 & \lfloor 19 / 4 \rfloor \\
3 & 4 \\
\end{array}
\]

Golomb codes are optimal for geometric distributions.

JPEG-LS uses the subfamily of Golomb power-of-2 (PO2) codes: $m = 2^k$

\[
G_{2^k} : n \rightarrow \left\{ (n \mod 2^k), \lfloor n / 2^k \rfloor \right\} \quad n \geq 0, \quad k = \text{code parameter}
\]

Encoding is very simple and explicit: no tables.

Very suited for adaptive coding: adapt $k$.
Adaptive Coding of TSGD’s in JPEG-LS

- Optimal prefix codes for TSGD’s are approximated in JPEG-LS by applying the Golomb-PO2 subfamily to a mapped error value:
  \[ \varepsilon \rightarrow M(\varepsilon) \text{ or } \varepsilon \rightarrow M(-1-\varepsilon) \]
  
  \[ 0,-1,+1,-2,+2, ..., \rightarrow 0, 1, 2, 3, 4, ..., \text{ (Rice mapping), or} \]
  
  \[ -1,0,-2,+1,-3, ..., \rightarrow 0, 1, 2, 3, 4, ... \]

- Adaptive code selection (parameter \( k \), mapping)
  
  Approximation of optimal strategy based on \textit{ML estimation} for TSGD parameters \( \theta, s \) through \textit{sufficient statistics}
  
  \[ A = \text{accumulated sum of error magnitudes} \]
  
  \[ N_- = \text{number of negative samples} \]

- Assumption \( s \in [0,1) \) satisfied through the use of \textit{adaptive correction} of the predictor, using also
  
  \[ B = \text{accumulated sum of error values} \]
  
  \[ N = \text{total number of samples} \]
Adaptive Coding (cont.)

q For the Golomb-PO2 code with parameter $k$ applied on remapped prediction errors

- compute the expected code length explicitly as a function of $\theta$, $s$, and $k$
- replace (the unknown) $\theta$ and $s$ by their ML estimates as a function of sufficient statistics

q Optimal decision regions for $k$ result in

- Let $\phi = (\sqrt{5} + 1)/2 \approx 1.618$ (golden ratio)
- If $A - N_\leq N \phi$, use $N_\leq$ to
  - choose $k = 0$ or $k = 1$
  - choose $s \geq \frac{1}{2}$ or $s < \frac{1}{2}$ if $k = 0$ (irrelevant if $k > 0$)
- If $A - N_\geq N \phi$, choose $k$ such that
  \[
  \frac{1}{\phi^{2^{-k+2}} - 1} < \frac{A - N_\leq}{N} \leq \frac{1}{\phi^{2^{-k+1}} - 1}
  \]
  \[
  k > 1
  \]
Approximation of Decision Regions

q Observe \( \varphi^{2^{-k}} - 1 \approx 2^{-k} \ln \varphi \)
\[ \ln \varphi \approx 0.48 \approx 0.5 \]
\[ \Rightarrow \frac{1}{\varphi^{2^{-k}} - 1} + 0.5 \quad \text{is always close to a power of } 2 \]

q Since \( N_- / N \approx 0.5 \), \( k \) can be approximated by \( k \approx \left\lfloor \log_2 (A/N) \right\rfloor \)

Estimate \( k \) using the trivial loop
\[
\text{for } (k=0; (N<<k) < A; k++) ;
\]

- If \( k=0 \) and \( s < -1/2 \), encode \( M(-\varepsilon-1) \), otherwise \( M(\varepsilon) \)
- Only 4 variables per context are needed
Embedded Run-length Coding

- Aimed at overcoming the basic limitation of 1 bit/pixel inherent to pixel-wise prefix codes, most damaging in low-entropy regions.

- Creates super-symbols representing runs of the same pixel value in the “flat region” \( a = b = c = d \) \( \Rightarrow \) special context \([q_1,q_2,q_3]=[0,0,0]\)

- A run of \( a \) is counted and the count is encoded using block-MELCODE, a fast adaptation technique for Golomb-type codes.
LOCO-I in One Page

loop:
q Get context pixels \( a, b, c, d \), next pixel \( x \)
q Compute gradients \( d-b, b-c, c-a \) and quantize \( \Rightarrow [q_1, q_2, q_3, \text{sign}] \)
q \([q_1, q_2, q_3] = 0 ? \) YES: Process run state NO: Continue
q \( x_{\text{pred}} = \text{predict}(a, b, c) \)
q Update correction value for context. Correct \( x_{\text{pred}} \)
q \( \varepsilon = x - x_{\text{pred}} \). If \( \text{sign} < 0 \) then \( \varepsilon = -\varepsilon \)
q Estimate \( k \) for the context
q Remap \( \varepsilon \rightarrow M(\varepsilon) \) or \( \varepsilon \rightarrow M(-1-\varepsilon) \)
q Encode \( M \) with Golomb-PO2(\( k \))
q Back to loop

run state:
q Count run of \( a \) until \( x \neq a \Rightarrow \) run length \( L \)
q Encode \( L \) using block-MELCODE
q Update MELCODE state
Compression/Complexity trade-off

Relative running time (approx.)

Average bits/symbol on benchmark image set

- CALIC-A
- JSLUG
- ALCM
- CALIC-H
- LOCO-I
- JPEG-A
- FELICS
- JPEG-H