AT&T Bell Laboratories

subject: **Natural versus "Universal" Probability, Complexity, and Entropy**
Work Project No. 311306-0399
File Case 38543

date: **23 November 1992**

from: **John S. Denker**
Dept. 11389
HO 4g330
908 949 8128
jsd@neural.att.com

Yann leCun
Dept. 11389
HO 4g332
908 949 4038
yann@neural.att.com
11389-921023-26TM

TECHNICAL MEMORANDUM

The interrelated concepts of *information, entropy, probability,* and *universal computation* enjoy very wide application It is worth noting, though, that the successful applications and theorems are mostly restricted to systems having a vast number of particles or symbols For smaller systems, it is not clear how to quantitfy the information/entropy/probability precisely. We argue that these are not just vague, academic questions, but sometimes definite problems in practice.

Mathematics and computer science allow us to construct any number of inequivalent probability measures. Even so-called "universal" probability measures can be quite inequivalent. Deciding which of these best describes the real world is a task for empirical science. As illustrations we consider the (1) information content of the genome and (2) a practical "learning from examples" task.

Additional keywords: Universal Turing Machine, Universal Data Compression, Universal Prior, Algorithmic Complexity, Kolmogorov Complexity, Machine Learning.

1 Introduction

We celebrate the interrelated concepts of "entropy," "information content," "universal computer," and "probability" as being among the greatest achievements of modern science. Despite (or perhaps because of) their sacredness, we ought to examine their foundations. Doing so reveals some surprises. For instance: (1) While there is no problem measuring precisely the entropy of a macroscopic system, the situation is not so clear for a microscopic system (2) Certain data-compression schemes are claimed to be optimal in the sense that they work essentially as well as any other, if not better. In fact, though, compressor "A" might work better than compressor "B" when applied to application "a," and vice versa when applied to application "b." The performance difference may well disappear asymptotically when very long data-strings are considered, but the asymptotic performance is not

generally a good predictor of the typical case. (3) Similarly, certain learning algorithms are claimed to be optimal in the sense that they work essentially as well as any other, if not better. But once again, they do so only in the limit of infinite amounts of training data, while practical applications involve distinctly finite amounts of data

The literature abounds with implicit and explicit attempts to use computational complexity to derive "the" unique probability distribution describing everything in the real world. We will argue that these attempts are misguided. Consider the analogy: in the days of Euclid it was thought that pure mathematics could uncover axioms that describe the geometry of the universe. More modern mathematics, though, allows us to discuss lots of different geometries. Nowadays nobody thinks that the metric of real spacetime is uniquely, axiomatically defined by pure mathematics; discovering the "real" geometry is a task for empirical science.

The analogy to probability theory is this: modern mathematics allows us to discuss lots of different probability measures. Determining which of these actually describes the real world is not a fit subject for mathematics, or for abstract computer science.

2 Basic Properties of Algorithmic Complexity

Algorithmic complexity[1, 2, 3] is based on the notion that a long string of symbols exhibiting an easily-described pattern is not random, while a long string of symbols not exhibiting any identifiable pattern ought to be called random. (This notion of "randomness" also goes by the name of "complexity," and the opposite of "complex" is "simple"; for a review, see [4] .) Extreme cases include

(a) the string consisting of a million zeros, which is very, very simple;
(b) the string consisting of "01" repeated 500,000 times, which is almost as simple; and
(c) the string generated by tossing a coin a million times, which is quite complex.

The crucial intermediate case is exemplified by

(d) the string representing the first million digits of π.

The digits individually would pass almost any test for randomness and independence, but in some sense this is hardly more complex than string (a) or (b), since each digit is completely predictable.

These ideas can be formalized as follows: the algorithmic complexity K of a symbol-string S is defined to be the length (generally measured in bits) of the shortest program on a given computer [5] $c1$ that will emit that string and then halt. In symbols:

$$K_{c1}(S) := \min_P \{\mathrm{Len}(P) \mid \mathrm{eval}(c1, P) = S\} \tag{1}$$

where $\mathrm{eval}(c1, P)$ denotes the result of running program P on computer $c1$.

In addition to providing a precise measure of complexity, $K_{c1}(S)$ has many remarkable properties. In particular, it makes contact with the axioms of probability (as will be discussed below), and hence with the idea of entropy, with endless ramifications in communication theory[6] , physics[7] , and machine learning [1] .

If we have another computer $c2$ available, it will generally assign a different complexity to the given string S. Kolmogorov demonstrated the remarkable fact that for any computer $c2$ and any universal computer c1,

$$K_{c1}(S) \leq K_{c2}(S) + k12 \tag{2}$$

where $k12$ depends on $c1$ and $c2$ but does not depend on S; i.e. a value of $k12$ can be found which makes equation 2 hold for all strings S. Basically $k12$ is the length of the "emulator program" $\mathrm{Em}(c1|c2)$ which enables $c2$ to emulate $c1$. There is no guarantee that $k12$ will be small. Henceforth

we assume $c2$ is also universal, by using equation 2 twice we get two-sided (upper and lower) bounds on the complexity difference between any two universal machines

If we consider the limit of very complex strings S, the *ratio* K_{c1}/K_{c2} goes to unity:

$$\lim_{K_{c1}(S)\to\infty} \frac{K_{c1}}{K_{c2}} = 1 \tag{3}$$

In this sense the constant $k12$ becomes negligible and K loses its dependence on the choice of computer $c1$ or $c2$ This is the basis for the claims that K is an all-purpose measure of complexity On the other hand, whenever we wish to compute the *difference* $K_{c1}-K_{c2}$, the constant cannot be neglected

One often hears the emphatic statement that "K is universal" This statement is correct in a technical sense, because the literature in this field defines the word "universal" to denote the property of "differing by at most a constant" as exhibited in equation 2. The problem is, this technical definition diverges quite a bit from the vernacular meaning of "universal."

Although K is "universal" in this technical sense, it has not been proven that K is all-purpose, or general-purpose, or universal in the vernacular sense. In particular it is an abuse of language to speak of "the universal distribution" when there are in fact many non-identical "universal" distributions.

Equivalence

Things that are equivalent in one respect need not be identical in all respects. For instance, "same-colored objects" constitute an equivalence class, but we cannot conclude that all red objects are the same size. The "universal" complexity measures form an equivalence class, when compared in the limit as in equation 3 — but that does not mean they are equivalent for all purposes.

It is often argued that the asymptotic behavior (equation 3) is a guide to the finite-case behavior. Our point, as illustrated below, is that such guidance is not 100% reliable.

Application to Communication Theory

In communication theory, it is often desired to calculate the average cost per bit of transmitting a message For simple strings, i e. where the length of the program for S is smaller than the length of S itself, we could save money by transmitting the program instead of the string. This is called algorithmic data compression. It makes sense to compare the efficiency of two coding schemes by taking the ratio of their code lengths. In this specialized application, and in the limit of long transmissions (i.e complex strings), it is permissible to forget what computer is used to evaluate the complexity — it doesn't matter since the length of any emulator can be amortized over a large number of message bits. For finite strings, however, $K_{c1}()$ is certainly has no unique value (since it depends on $c1$) and is not provably an optimal (or even a good) coding scheme. Indeed, we believe it is unlikely to be good, for "typical" data streams and "typical" computers. Another blemish is that Kolmogorov complexity is noncomputable[15] — the definition (equation 1) runs afoul of the halting problem

3 The Connection between Complexity and Probability

There is a deep connection between complexity and probability. We define the following probability distribution over strings.

$$P_{c1}(S) := \lim_{L\to\infty} 2^{-L} \sum_{P} \delta^{S}_{\mathrm{eval}(c1,P)}\, \delta^{L}_{\mathrm{Len}(P)} \tag{4}$$

where δ is the Kronecker delta. This is equal to the probability that a randomly-tossed program would generated S and then halt. To understand how the limit converges, consider the program P_1 that is the shortest program that emits S; it has length $K_{c1}(S)$. There will 2^N ways of padding P_1 to length $K_{c1}(S) + N$ using all possible suffixes. Therefore for each $L \geq K_{c1}(S)$, the contribution of P_1's daughters to $P_{c1}(S)$ will be constant. If there are alternative ways of computing S, using programs for which P_1 is not a prefix, $P_{c1}(S)$ will exceed $2^{-K_{c1}(S)}$. Usually these alternatives do not contribute significantly, so the negative-log-probability is essentially equal to the complexity. Indeed, it is arguably a better definition of complexity.

By extension, for a set of strings σ, we define

$$P_{c1}(\sigma) := \sum_{S\in\sigma} P_{c1}(S) \tag{5}$$

The function P_{c1} maps sets (of strings) into positive numbers, and the P_{c1}-value of the union of disjoint sets is the sum of the P_{c1}-values of the ingredients. Therefore it is a measure, by definition of measure. Furthermore, since it is bounded above (by unity, if not less), it is a probability measure, by definition of probability measure

Application to Statistical Physics

Physicists commonly have a keen and robust intuition about probability. A common belief is that there is "one true probability distribution" that characterizes the universe; theory are experiment serve to discover that that distribution is. Much of the the Bayesian inference literature also seems to adhere to this belief. This belief is sometimes so strong that people are unwilling even to consider the notion of more than one probability distribution. However, in the spirit of multiple non-Euclidean geometries, we must now at least talk about multiple probabilities — perhaps later we can decide which one describes Nature.

This yields a simpler yet more sophisticated way of understanding the recent interesting work of Zurek[7] , in which he rederived the foundations of physics using 2^{-K} "instead" of probability. But we see that it cannot be considered a substitute for probability — it must be considered a probability, no more, no less, according to the axioms of probability theory.

We suspect that "a" probability distribution derived from an ordinary computer or Turing machine will not in fact describe well "the" probability distribution of the real universe.

Not an All-Purpose Probability

When complexity is used to compute a probability via formula 4, it generally does not make sense to take the ratio of complexities. It makes much more sense to compute the ratio of probabilities, but that corresponds to the *difference* of complexities. The constant $k12$ must not be neglected.

To gain intuition about the consequences of modest values of $k12$, consider two computers $c1$ and $c2$ which are identical except that the latter requires an N-bit password at the head of all programs. Then for any string S_i,

$$P_{c2}(S_i) = 2^{-N} P_{c1}(S_i) + (1 - 2^{-N}) P_{c1}(S_0) \tag{6}$$

where S_0 is the string "Permission denied" that $c2$ emits when the required password is absent. If the password length is, say, $N = 20$ bits, we see that for all strings other than S_0, $c2$ assigns a probability a on times smaller than $c1$ would. This ratio persists even for very complex strings. Twenty is not very many, but million-fold misjudgements of probabilities are often quite significant in actice Therefore the distinctions that lie behind $k12$ must be taken very seriously.

It may seem inconsequential to change the probability of all strings (or all except one) by a constant, therefore consider the following elaboration of the previous example. Imagine a computer $c2$ that calls $c1$ as a "subroutine." It can examine $c1$'s output string before deciding whether to enforce a password requirement on its program. In this way $c2$ can change the probability of any computable subset of the set of all strings.

To put this in mathematical terms, we conclude that the convergence of the limit in equation 3 is highly non-uniform. Consider the two statements

- For any two universal computers $c1$ and $c2$, there exists some F such that for all strings S, $P_{c1}(S)$ differs from $P_{c2}(S)$ by at most a factor of F
- On the other hand, for any universal computer $c1$ and for any factor $F \geq 1$, there exists another computer $c2$ such that $P_{c1}(S)$ differs from $P_{c2}(S)$ by more than F on almost all strings S.

As mentioned above, it is often argued that the asymptotic behavior can be used a guide to the finite-case behavior One should be very careful, though, when trying to generalize a result that depends sensitively on the order of limits.

4 Un-Algorithmic Complexities

The power of the computer gives us the power to create very peculiar probability distributions, but we must dispell the notion that it is somehow more powerful than other ways of specifying a distribution; in fact it is less powerful. As a simple yet practical example, let the probability of a particular string S_0 be zero, corresponding to infinite complexity Modeling this distribution would require a computer $c0$ that is clearly not a universal computer.

As a more troubling example, consider the un-algorithmic probability distribution

$$P_{c3}(S) = 2^{-\alpha K_{c1}(S)} \quad \text{for some } \alpha > 1 \tag{7}$$

for all $S \neq S_0$, where $c1$ is some universal computer (and if you want, you can assign S_0 enough probability to make P_{c3} normalized) The distribution P_{c3} is perfectly reasonable and well defined; it just assigns a lower probability to long strings than any distribution based on a universal computer[12] would. (Otherwise the proportional relationship in equation 7 would violate the bound in equation 2) The point remains· there exist perfectly reasonable probability distributions that cannot be based on algorithmic complexity in a reasonable way.

We note in passing that Chaitin's remarkable number Ω [11] has no unique value; its value and meaning are tied to a particular choice of computer.

Similarly, the "universal" Lempel-Ziv coding scheme[16] just ascribes an arbitrary prior probability to strings Their probability works quite well for compressing ordinary text, but not for other strings, e g rasterized pictures. It is "universal" in the technical, asymptotic sense, but is not unique or all-purpose or provably optimal in any practical sense.

Also similarly, Rissanen's "universal prior for integers"[13] has no particular claim to unique correctness. There are innumerable ways of dividing the available probability measure among all the integers. Indeed, consider any non-negative function $F(i)$ for which the sum $C = \sum_{i=-\infty}^{\infty} F(i)$

exists, and assign $P(i) := F(i)/C$. Any F-function that is strictly positive (i.e. nonzero) is universal in the technical sense. There are infinitely many choices for F; the "right" choice is highly application-dependent. Rissanen's proposed prior just constitutes a particular choice of F.

Assigning probabilities to strings is no different from assigning probabilities to integers; there are countably many strings and they can be placed in one-to-one correspondence with the integers.

The most general way of assigning probabilities is to make a long list with two columns, writing down each string next to its corresponding probability. This is just a low-tech way of specifying the function $F(i)$. If there is a more concise specification, such as $F(i) = \exp(-i^2)$, then we have the remarkable possibility of assigning probabilities to countably many strings using only a finite amount of specification.

The function $F(i)$ can be considered a computer program for computing F given i, but that does *not* mean that this formulation is equivalent to computational complexity; for instance F could be zero, or could implement the un-algorithmic probability of equation 7.

5 Absolute Descriptions

In any case, we face the question of how many bits it takes to describe a given computer. Suppose $c1$ is a particular universal Turing machine. As explained above, it will not do to say "all universal Turing machines are equivalent" — they may be equivalent with respect to *computability*, but they are not equivalent with respect to computational complexity.

As mentioned above, $k12$ can be thought of as the length of the emulator program that allows $c2$ to emulate $c1$ We could, in turn, have another computer $c3$ that emulates $c2$, and so forth; $k12 + k23$ would be a bound on $k13$ This corresponds roughly, but not exactly, to the chain property of conditional probabilities:

$$P(a) = P(a|b)\ P(b) \tag{8}$$

Describing one computer in the language of another is not particularly helpful unless the process terminates at some point in an absolute, primordial computer that does not need describing.

The structure of a Turing machine can be specified in detail using a bounded amount of English textual description (although it might be easier to use a graphical blueprint). English is a higher-level language for a common "computer" that uses neurons for the machinery. The structure of this computer, the human brain, is specified by a bounded amount of DNA (plus a bounded amount of data acquired by learning)

6 Information Content of the Genome; Natural Probability Distributions

As an illustrative application of the foregoing ideas, we will now argue that the information content of the genome poses nontrivial issues of complexity and probability. Some of the arguments in this section parallel the arguments in the following section.

In vertebrates, the retina and the visual processing centers of the brain arise separately, and get "wired up" relatively late in embryonic life The connection pattern is not trivial; drugs or disease can cause the mapping to become a random permutation, with serious consequences.

If one were to make the seemingly reasonable assumption that all possible permutations were *a priori* equally probable, the correct connection pattern would be vanishingly improbable. The creature would not be able to learn the correct permutation, since no lifetime is long enough to gather the

required information What's worse, under these assumptions the correct permutation could not be inherited, since to specify the pattern would require more information than is contained in the creature's entire genome.

Hence the "uninformative" prior that all permutations are equally likely is unacceptable. The probabilities and the information budget must be computed using computational complexity. As established in the previous sections, we must be careful to evaluate the computational complexity using the "right" computer The question is, what is "right" in this case?

The machine language of the "natural" computer is determined by biophysics and biochemistry. The creature's DNA is a program that runs on this computer. In our example, the actual retinotopic connection "procedure" is known to use concentration gradients of certain marker molecules diffusing in physical space to preserve topological properties: the idea is that neighboring parts of the retina map to neighboring parts of visual cortex. There are very few permutations that preserve topology, compared to general permutations. We conclude that topology-preserving connections are probable because they are simple to express in the language of the biological computer.

Some people have tried to understand probabilities in terms of systems of plausible belief. This view cannot encompass physical processes such as embryogenesis — the complexity of the the optic nerve connection does not depend on what anybody believes about it.

The brain needs to understand the physical world. The brain uses, internally, the laws of physics to build models of the external world. When considering the information content of the genome, it is crucial to realize that the genome need not contain a specification of the topology and dimensionality of the universe — they can be taken as "given" — on the right-hand side of the "given" bar in all conditional probabilities.

To quote Walt Whitman, "A man is a piece of the universe made alive."

The architecture of the biological computer stands in stark contrast to the architecture of typical 1990s computers, which go to some trouble to conceal the topology and dimensionality of the universe in which they reside. A "good" computer tries to make every word in memory equally easy to access. In a typical workstation, a retinotopic map is no easier to construct when it represents 2 dimensions, as opposed to 11 or 26 or any other unphysical ·–) number of dimensions.

Computer scientists are fond of saying that "the structure of the computer does not affect the structure of the computation" That is an overstatement. The structure of the computer may not affect questions of computability, but it does affect computational complexity.

7 Connection to Learning

There is a deep connection between learning and compression[14] .

Learning depends on both (a) prior knowledge and (b) training data. The role of prior knowledge is infinitely important (although it is often not discussed) In the discrete case, a simple counting argument[8] shows that without an informative prior, automatic learning cannot result in useful generalization In the case of a more general hypothesis space, the work of Vapnik & Chervonenkis [9] yields the corresponding result in order to learn from examples and form a reliable generalization, the number of training examples must scale like the "richness" of the hypotheses space. This idea can be made quite precise

In general, ignorance plus statistics equals ignorance.

As a concrete example, consider the Optical Character Recognition (OCR) system that was built in our lab[10] . The design and training of this system can be viewed as a process of progressive restriction At each step in the process, the amount of information necessary to specify a working system (the number of free parameters) is much less than at the previous step. Specifically:

1. Neural Network — We start with the assumption that the task could be performed by an artificial neural network with a limited number of processing units. This is already quite a strict assumption; although it could be motivated by the fact that humans can perform the task using supposedly analogous resources, the true justification for our assumptions is that the final system actually works
2 Feed-Forward, Layered Network — The most general neural network has every unit connected to every other unit, but we restricted our attention to networks in which the connection pattern is a directed acyclic graph, with at most five steps from input to output.
3 Sparse Network — We further restricted the network so that in each layer (except the last), the number of connections was very small compared to the number of possible connections.
4. Local Network — We assigned X and Y labels to the units in each layer (except the last) in a way that imputed to them a position in a two-dimensional array. We then assured that each unit's incoming connections came only from units "close" (in the two-dimensional sense) to the the corresponding unit in the previous layer. Once again this connection pattern is a restriction of the previously-mentioned connection pattern.
5. Convolutional Network — We further restricted the network so that the pattern of incoming connection strengths was the same for all units in a given layer; that is, the connection pattern was invariant to shifts in X and Y.
6 Trained Network — Finally, we trained the network. That is, we used "BackProp"[8] to search for a set of connection strengths that were (in addition to the foregoing restrictions) maximally consistent with a training set containing many thousands of images.

It is instructive to consider what would happen if our design process had omitted restrictions 3 (sparseness) 4 (locality) and 5 (shift invariance). In that case, the X and Y labels assigned to the units would be dummy indices — units could be permuted without changing the connection pattern, since everything is connected to everything else (within layers). One consequence is that such a network would be invariant under permutation of the input pixels. Imagine how hard it would be for a human to learn the required OCR task by viewing patterns through a fiber-optic system that scrambled the incoming images. As a general rule, if you have information about what pixels are neighbors of other pixels, you should use that information You don't want to design a system that is invariant under encryption

To put it another way, the purpose of the restrictions on the network was to "program in" a very important piece of information. The OCR images are not random collections of ones and zeros — they come from images on real, two-dimensional paper. Therefore we constructed a specialized computer (i.e. the neural network) in which the short programs correspond to functions likely to be useful in solving our problem The network should not have to guess the dimensionality of its input data

8 Conclusions

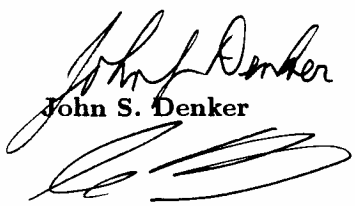
Algorithmic complexity has heretofore been used primarily to measure the asymptotic complexity of very long strings. For such applications, it may not be necessary to specify the computer used to determine the complexity. The term "universal" has been used in this context, by dint of a highly technical definition; one should not imagine that such a complexity measure is all-purpose, unique, or "universal" in the vernacular sense

The structure of the computer generally does not affect questions of computability, but it strongly affects computational complexity.

A perceptual system should not be invariant under encryption. An adaptive system should not need to learn or guess the dimensionality or symmetry of the world in which it operates. Ignorance plus statistics equals ignorance.

There are many situations where knowing the right prior probability is important, including learning from examples, data compression, and statistical physics. There are many possible inequivalent probability distributions. Finding the right one for a given application is not a task for abstract mathematics or computer science; it depends on observable properties of the physical world.

Notions of complexity are attracting mounting attention and are being extended to new areas. We must take care that these extensions are not burdened by misconceptions based on no-longer-valid side conditions or intuition



John S. Denker

HO-11389-jsd/YlC

Yann leCun

References

[1] R. J. Solomonoff, "A formal theory of inductive inference" (Part I and Part II), *Information and Control* **7**, pp. 1–22 and 224–254 (1964).

[2] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Problems in Information Transmission* **1**(1) pp 1–7 (1965).

[3] G J. Chaitin, "On the length of programs for computing finite binary sequences," *J. Assoc. Comp. Mach.* **13** pp. 547–569 (1966).

[4] M. Li and P. M B. Vitanyi, "Two decades of applied Kolmogorov complexity: in memoriam Andrei Nikolaevich Kolmogorov 1903-87," in proceedings of the Structure in Complexity Theory Third Annual Conference, pp. 80-101, IEEE Comput. Soc. Press, (1988)

[5] In the interests of clarity, we have brushed aside some niceties, such as the distinction between a computer (examples of which actually exist) and a universal Turing machine (which is an abstraction); formal definitions and proofs would use the latter.

[6] T. M. Cover and J. A. Thomas, *Elements of Information Theory* Wiley-Interscience, New York (1991)

[7] W. H. Zurek, "Thermodynamic cost of computation, algorithmic complexity and the information metric," *Nature* **341**(6238), pp. 112–124 (September 1989)

[8] J S. Denker et al., "Automatic Learning, Rule Extraction, and Generalization," *Complex Systems* **1**, pp 877–922 (October 1987).

[9] V. N. Vapnik and A. Ya Chervonenkis, "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," *Th. Prob. and its Applications* **17**(2) pp 264–280 (1971).

[10] Y. Le Cun, B. Boser, J S. Denker, D Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Back-Propagation Applied to Handwritten Zipcode Recognition," *Neural Computation* **1**(4) (December 1989).

[11] G. J. Chaitin, *Algorithmic Information Theory*, Cambridge University Press (1987).

[12] It is possible to cobble up a *non-universal* computer $c3$ that would have program lengths $K_{c3} \sim \alpha K_{c1}$, but we see no advantage in doing so. This example is more troubling than the previous example because $c3$ is "almost" universal — any string that could be emitted by $c1$ could (eventually) be emitted by $c3$.

[13] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*, World Scientific (1989).

[14] It has been proved that you can do learning if and only if you can do compression. Basically, this is a consequence of the definitions: Imagine a sequence of symbols. After learning from the first N symbols, the task of the learner is to predict symbols $N+1$ through $N+M$. Similarly, the task of the compressor is to emit a short code for symbols $N+1$ through $N+M$. If the compressor is effective, it must emit short codes often, and long codes very rarely. Argument #1 (compression-based learning): The learner should guess outcomes for the M test symbols with probability 2^{-L}, where L is the length of the compressor's code. If the compressor is very good, the learner will be right much more often than random, which implies successful learning. Converse argument (learning-based compression)· use the learner to predict the M test symbols Transmit a code that specifies what the errors are. Good learning implies very few bits transmitted.

[15] A M. Turing, "On computable numbers with an application to the Entscheidungsproblem," *Proc. London Math. Soc* **42** pp. 1134–1142 (1936); correction *ibid.* **43** pp. 544–546 (1937).

[16] J. Ziv and A Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inform Theory* **IT-23** pp 337–343 (1977).