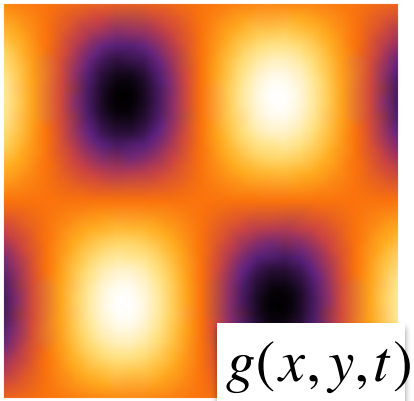
Image Data Compression

Introduction to Coding

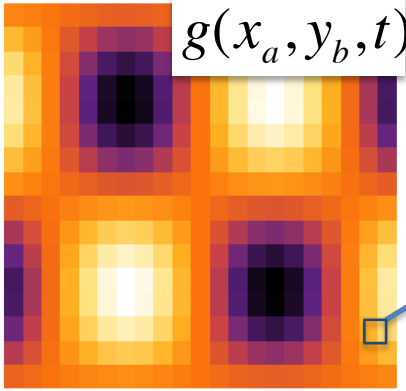
Review: data reduction steps (discretization / digitization)

Continuous 2D signal
(light intensity on sensor)



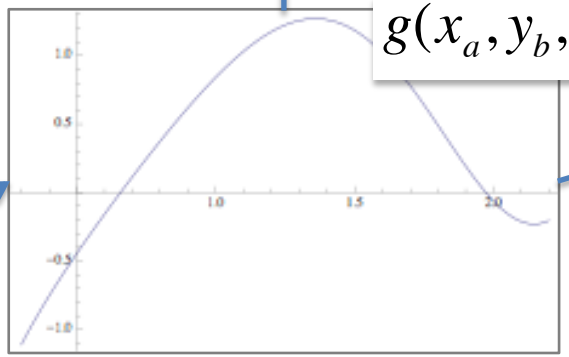
$g(x, y, t)$

Spatial discretization



$g(x_a, y_b, t)$

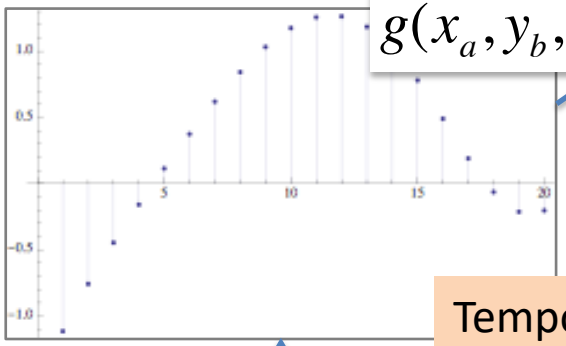
Spatially discrete signal
(pixel-averaged intensity)



$g(x_a, y_b, t)$

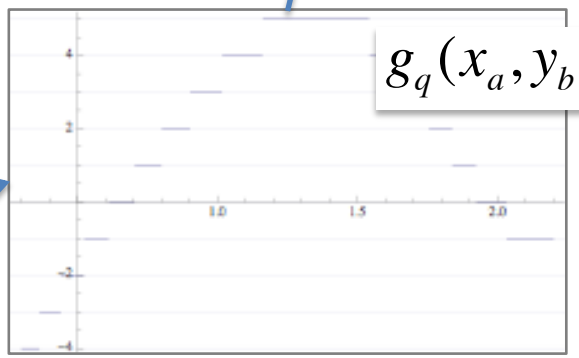
Analog signal
(light intensity at a pixel)

Discrete time signal
(pixel voltage readings)



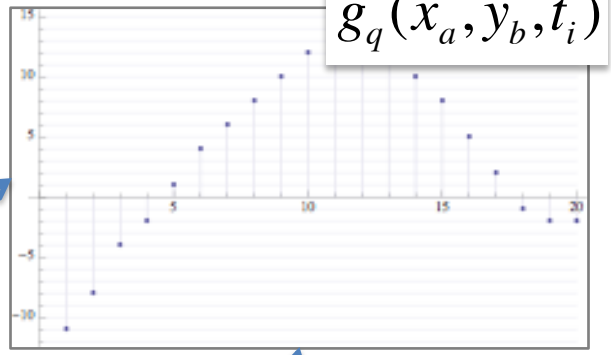
$g(x_a, y_b, t_i)$

Temporal discretization and digitization



$g_q(x_a, y_b, t)$

Discrete value signal
(e.g., # of electrons at each pixel of the CCD matrix)



$g_q(x_a, y_b, t_i)$

Fully digital signal

Review: data reduction steps (discretization / digitization)

Discretization of 1D continuous-time signals (sampling)

- Important signal transformations: up- and down-sampling
- Information-preserving down-sampling: rate determined based on signal bandwidth
- Fourier space allows simple interpretation of the effects due to decimation and interpolation (techniques of up-/down-sampling)

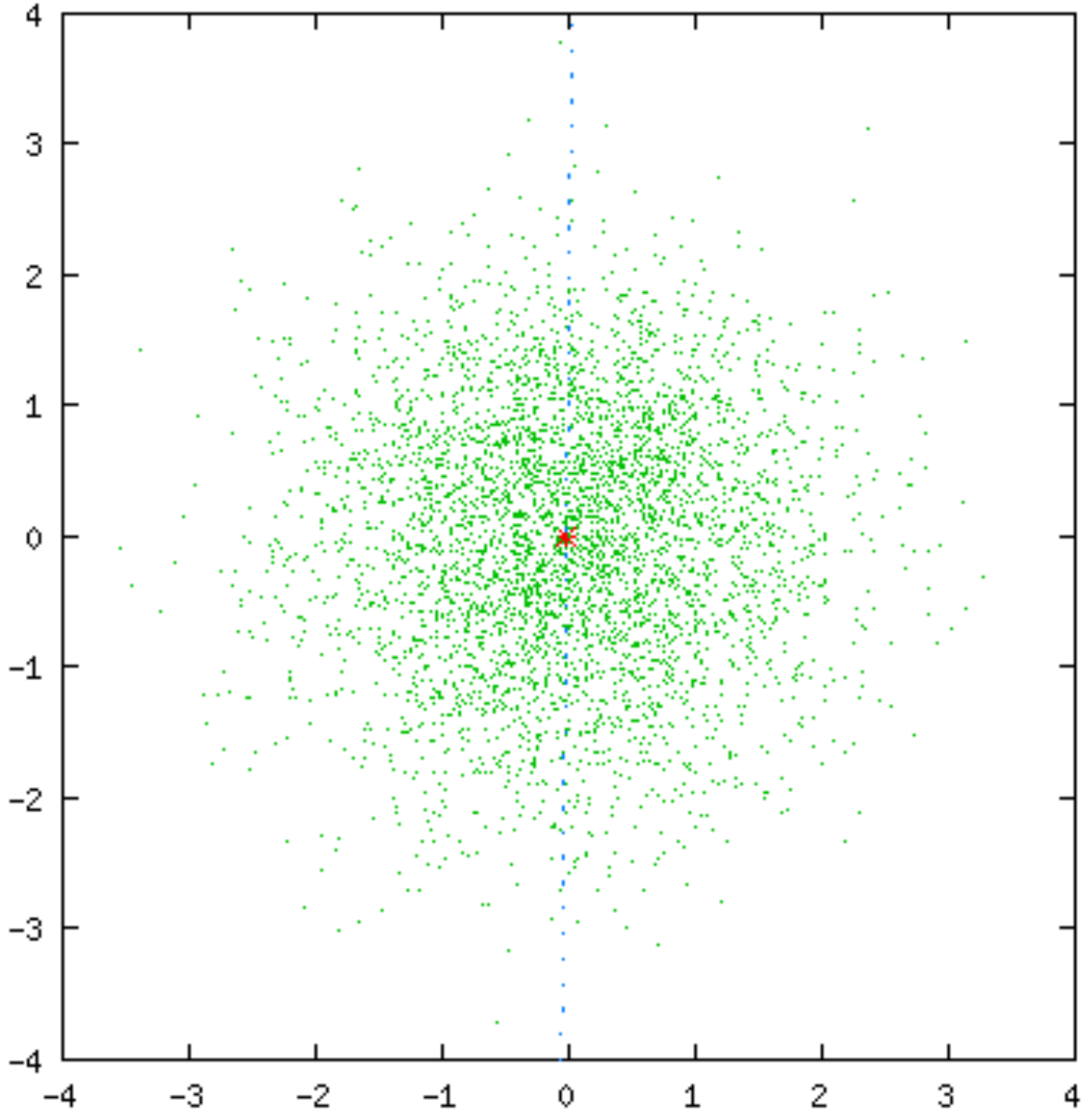
Scalar (one-dimensional) signal quantization of continuous-value signals

- Quantizer types: uniform, simple non-uniform (with a dead zone, with a limited amplitude)
- Advanced quantizers: PDF-optimized (Max-Lloyd algorithm), perception-optimized, SNR-optimized
- Implementation: pre-processing with a compander function + simple quantization

Vector (multi-dimensional) signal quantization

- Terminology: quantization, reconstruction, codebook, distance metric, Voronoi regions, space partitioning
- Relation to the general classification problem (from Machine Learning)
- Linde-Buzo-Gray algorithm of constructing (sub-optimal) codebooks (aka k-means)

LGB vector quantization – 2D example



[Linde, Buzo, Gray '80]:

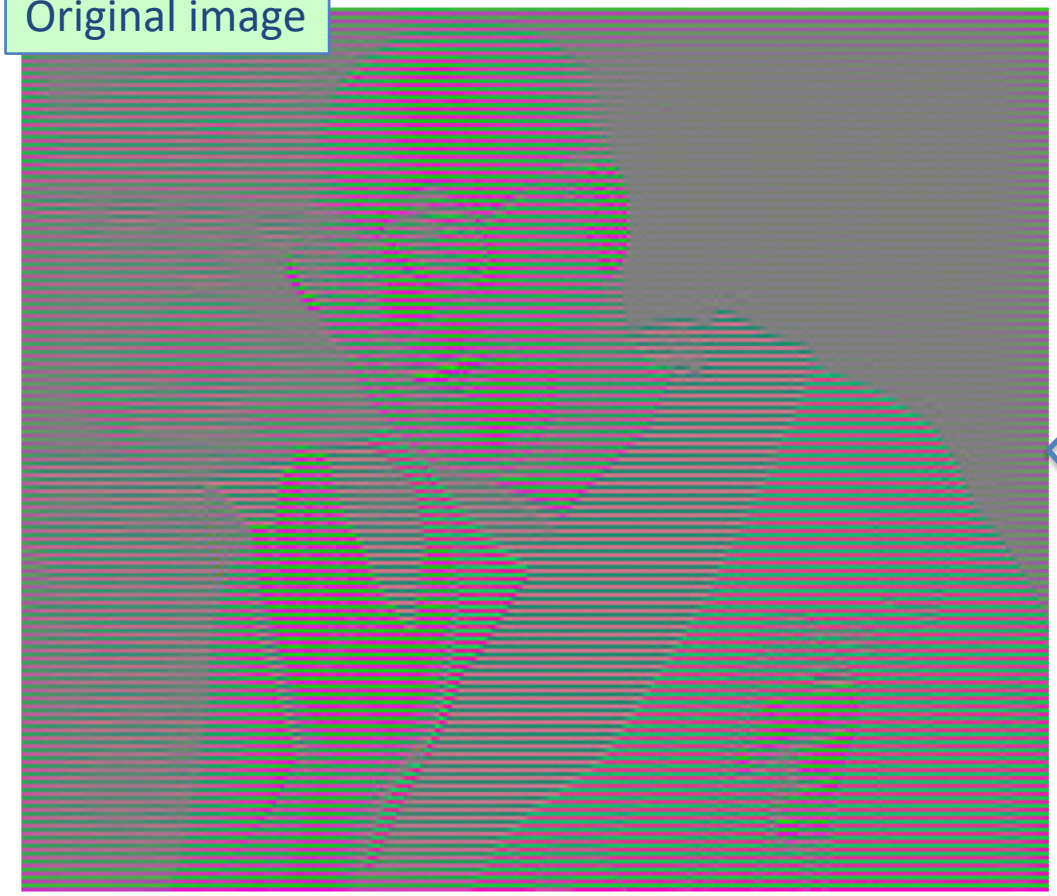
1. Choose codebook size N
2. Select N random codewords as the initial codebook Y
3. Classify training vectors (i.e. find V_i for each x_k)
4. Update codewords: in each cluster,

$$\vec{y}_i \leftarrow \frac{1}{N_i} \sum_{\vec{x}_k \in V_i} \vec{x}_k$$

5. Repeat steps 2 and 3 until convergence (i.e. until changes in codewords become small)

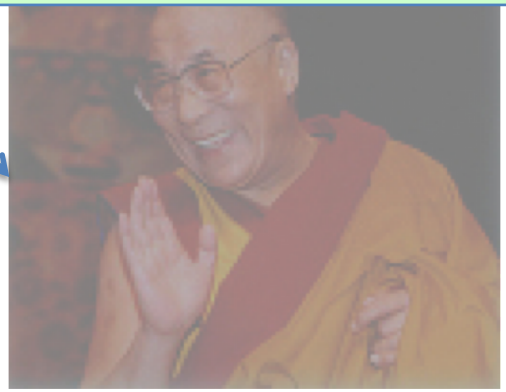
Example: gamma-error in picture scaling [Brasseur 2007]

Original image



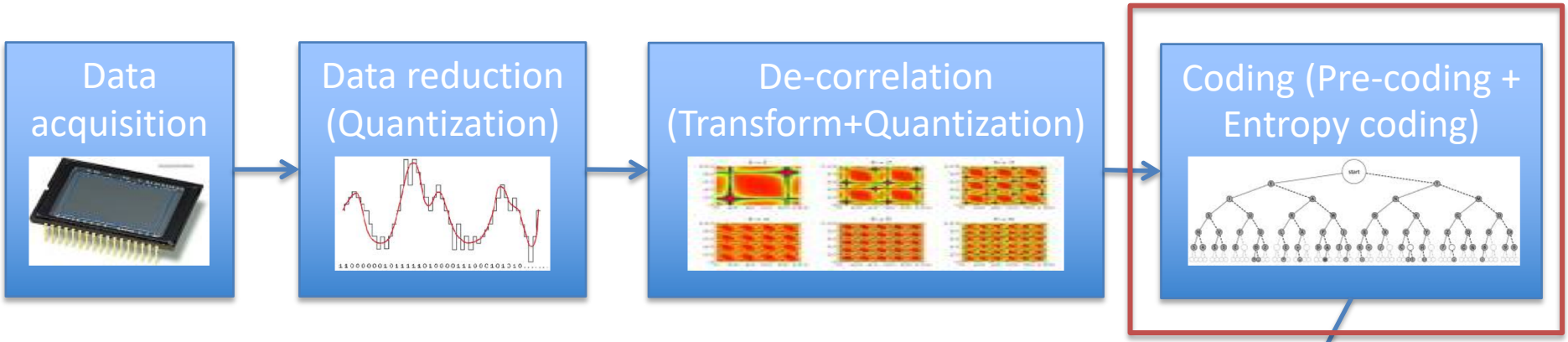
Resized image (averaging of indices)

Resized image (averaging of values)



Gamma-correction:

Image signal path: overview



Output data:
Continuous multi-dimensional analog signal

Compression:
Physical sensor limitations, DAQ settings

Output data:
Correlated fully digital multi-dimensional symbol stream

Compression:
Spatio-temporal quantization, value quantization

Output data:
Linear stream of de-correlated symbols

Compression:
Truncation of irrelevant signal parts, value quantization

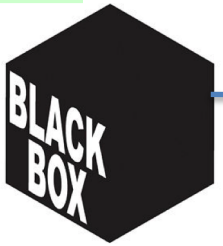
Output data:
Linear stream of bits

Compression:
Reversible suppression of the redundant information

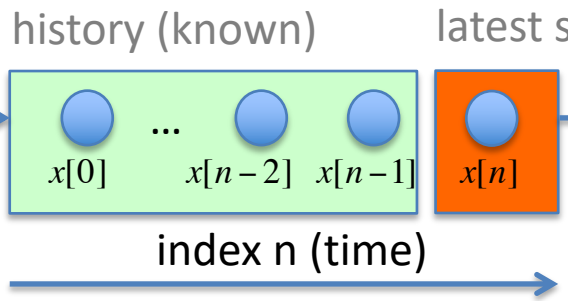
...

(Reversibly) coding finite strings: basic definitions

Source: produces a sequence of symbols



E.g., image data in some representation



Coder

(ideally, would like to code only the new information!)

Intuitively: how much new information does $x[n]$ bring?
It depends on how well we can predict $x[n]$, given the **a priori knowledge** and the available **history** ($x[0]$, …, $x[n-1]$).

Common notation

- Input **symbols** (characters, events): s_i
- Input **alphabet**: set of all possible distinct input symbols: $Z = \{s_i\}, \quad i = 1,\ldots,K$
- **Signal**: time-discrete finite sequence of symbols: $x[n] \in Z$, n is the time moment
  (Slight abuse of notation: $x[n]$ may denote the n-th value as well as the entire finite sequence $x[0]$ … $x[n]$)

- **Model**: probability that symbol s_i appears at the time moment n, given the entire signal history:

$$p_i^{(n)} \equiv p(x[n] = s_i \mid x[n-1],\ldots,x[0])$$

Coding finite strings: basic definitions (following [Shannon 1948])

Alphabet: $Z = \{s_i\},\ i = 1,\ldots,K$ **Signal:** $x[n] \in Z$ **Model:** $p_i^{(n)} \equiv p(x[n] = s_i \mid x[n-1],\ldots,x[0])$

[Ergodic] stochastic process, modeled as a Markov chain (a special type of Bayesian Networks):

0-th order model

$$p_i^n = 1/K$$

(no a priori knowledge)

Example: English, 27 letters: "xfoml rxkhrjffjuj zlpwcfwkcyj ffjeyvkcqsghyd qpaamkbzaacibzlhjqd"

1-st order model

$$p_i^{(n)} = p(x[n] = s_i) = p_i$$

(symbols are statistically independent, we account only for the frequency of individual symbols)

"ocroh hli rgwr nmielwis eu ll nbnesebya th eei alhenhttpa oobttva nah brl"

2-nd order model

$$p_i^{(n)} = p(x[n] = s_i \mid x[n-1])$$

(pairwise dependencies are important, need individual symbol frequencies and frequencies of digrams)

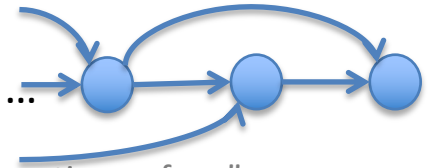
"on ie antsoutinys are t inctore st be s deamy achin d ilonasive tucoowe at teasonare fuso tizin andy tobe seace ctisbe"

3-rd order model

$$p_i^{(n)} = p(x[n] = s_i \mid x[n-1], x[n-2])$$

(dependence of each symbol on two previous symbols, i.e. add trigram statistics)

"in no ist lat whey cratict froure birs grocid pondenome of demonstures of the reptagin is regoactiona of cre"



...

Fully general model: infinite order (dependence of each symbol on all previous symbols)

Codeword-based coding of symbol strings: basic definitions

Alphabet: $Z = \{s_i\},\ i = 1,\ldots,K$ **Signal:** $x[n] \in Z$ **Model:** $p_i^{(n)} \equiv p(x[n] = s_i \mid x[n-1],\ldots,x[0])$

- Given a model, the **information content** of the actual symbol s_i is :
  ("the more surprising a symbol, the more information is brings")

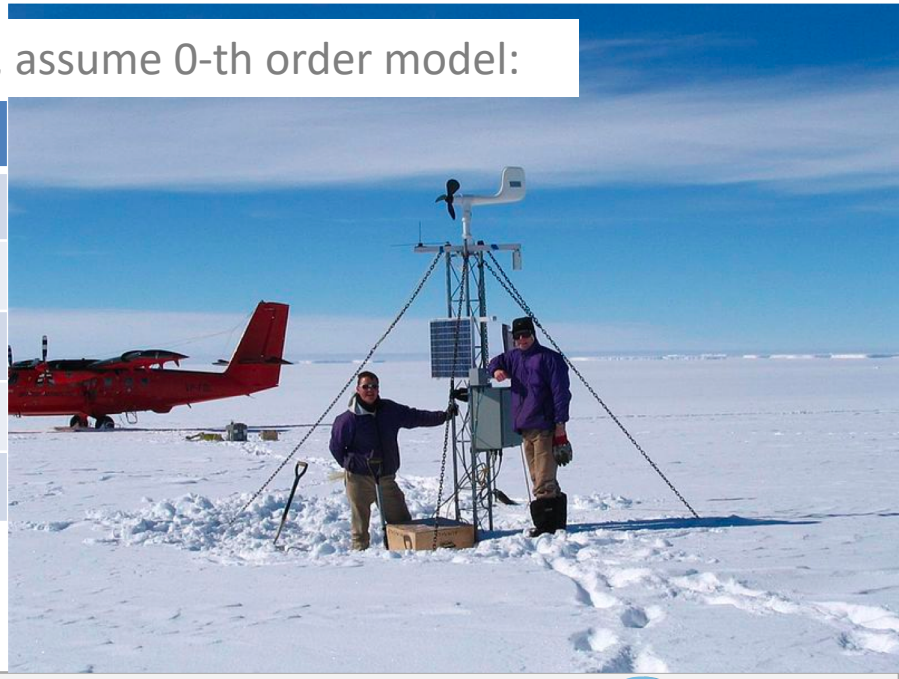
$$I(s_i) = -\log_2(p_i^{(n)}) \cdot [bit]$$

- **Codeword-based coding**: substitute each symbol with a **codeword**.
- Example: symbol **"A"** maps to the binary codeword **"01101"**, codeword length: 5 bits.
  (here we only discuss binary codes; relation to other bases can be easily established)

- **Code**: set of all codewords for an alphabet

Example: weather observations at a remote station, assume 0-th order model:

Weather	P = 1/K	I[bit]	Codeword
Sun	0.25	2	00
Clouds	0.25	2	01
Rain	0.25	2	10
Snow	0.25	2	11
	$\Sigma\ p_i = 1.0$		



Information content of each symbol matches its code length, no shorter representation

1-st order process model (entropy coding), Shannon bound

Alphabet: $Z = \{s_i\},\ i = 1,\ldots,K$ **Signal:** $x[n] \in Z$ **1-st order model:** $p_i^{(n)} = p_i$

- We always consider a "long enough" series of observations (n -> ∞)
- Assume an **ergodic source**: symbol statistics in a string approach those over all strings
- **Average codeword length**:

$$S_{src} = \langle l_i \rangle = \sum_{i=1}^{K} p_i \cdot l(s_i) \cdot [bit / symbol]$$

- **Source entropy**: average information content of a symbol (e.g. over any infinite string):

$$H_{src} = \langle I(s_i) \rangle_Z = \sum_{i=1}^{K} p_i \cdot I(s_i) = -\sum_{i=1}^{K} p_i \log_2(p_i) \cdot [bit / symbol]$$

- **[Shannon '48], [Shannon, Weaver '49]: average length of any entropy code is limited by H_{src}. No shorter code can be decoded unambiguously.** (We will not prove this theorem here.)

$$H_{src} \le S_{src}$$

- **Code redundancy**:

$$\Delta R = S_{src} - H_{src} \ge 0$$

Example: unary coding (assume a 0-th order model)

Alphabet: all decimal digits. A digit q is coded as “1” repeated q times and a terminal “0”:

Symbol	Code
0	0
1	10
2	110
3	1110
4	11110
5	111110
6	1111110
7	11111110
8	111111110
9	1111111110

Assume the equal probability of all ten digits:

$$p_i^{(n)} = p_i = \frac{1}{10}, i = 1,\ldots,10$$

Average codeword length for unary coding:

$$S_{src} = \langle l_i \rangle = \sum_{i=1}^{10} \frac{1}{10} \cdot (i+1) = 5.5$$

Source entropy:

$$H_{src} = \langle I(s_i) \rangle_Z = -\sum_{i=1}^{10} \frac{1}{10} \cdot \log_2\left(\frac{1}{10}\right) = 3.32$$

Code redundancy:

$$\Delta R = S_{src} - H_{src} = 2.17[bits/digit]$$

Example: binary coding (assume a 0-th order model)

Regular binary coding:

Symbol	Code
1	000
2	001
3	010
4	011
5	100
6	101
7	110

- Optimal if the alphabet size is a power of 2
- Wasted codeword length otherwise!
- (here: codeword “111” is not used)

Slightly more efficient: truncated binary coding

Given symbol d (d = 0, …, m - 1), compute

$$b = \lceil \log_2 m \rceil, \quad t = 2^b - m .$$

Then: if d < t, output (b - 1)-bit binary representation of d

Otherwise: output b-bit binary representation of (t + d)

- Can be decoded (longer codewords start with unused binary sequences)
- Optimal for the uniform probability distribution

$$H_{src} = 3.32[\text{bits/digit}]$$

$$S_{src} = 3.4[\text{bits/digit}]$$

b = 4

t = 6

m = 10

Symbol	Code
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0 0
7	1 1 0 1
8	1 1 1 0
9	1 1 1 1

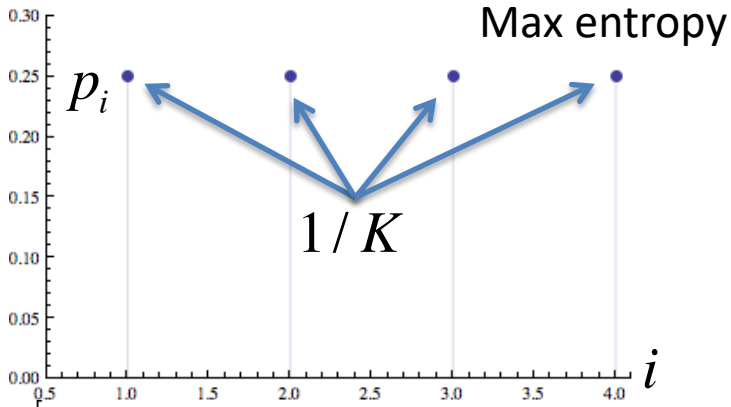
(b - 1) bits

b bits

Properties of entropy (discrete probabilities)

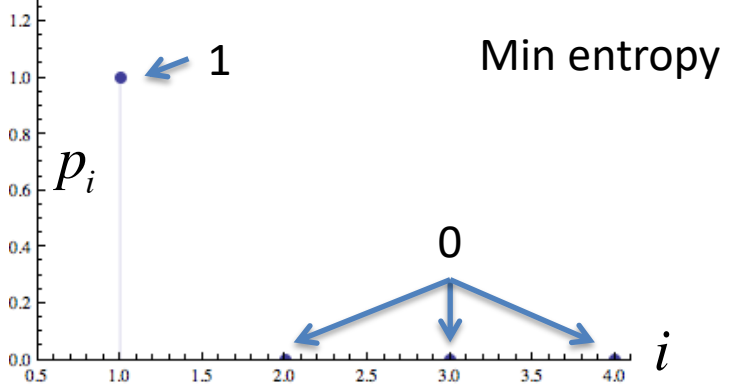
- Uniform distribution has the maximal entropy:

$$0 \le H \le H_{\max} = \sum_{i=1}^{K} \frac{1}{K} \log_2 K = \log_2 K$$



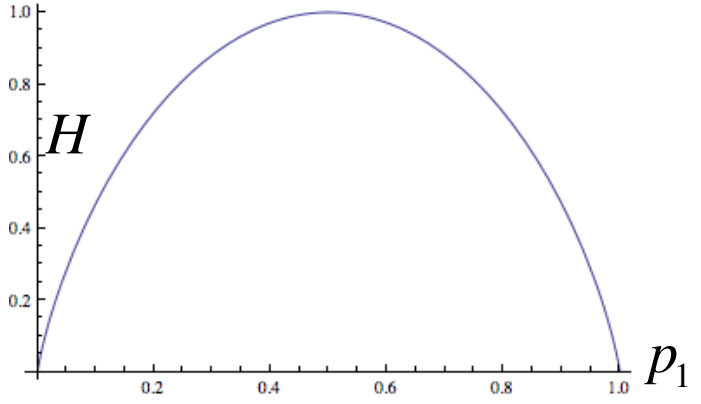
- If the next incoming symbol can be predicted (complete deterministic knowledge), one symbol has 100% probability, and the source entropy is zero:

$$H = 0 \Leftrightarrow p_i = \delta_i^j$$



- Special case: binary alphabet (two symbols):

$$H = -p_1 \log_2 p_1 - (1 - p_1) \log_2 (1 - p_1)$$



In other words, entropy is a measure of uncertainty

Entropy coding: process models of the 1-st order

Example: a 1-st order model, weather observation data:

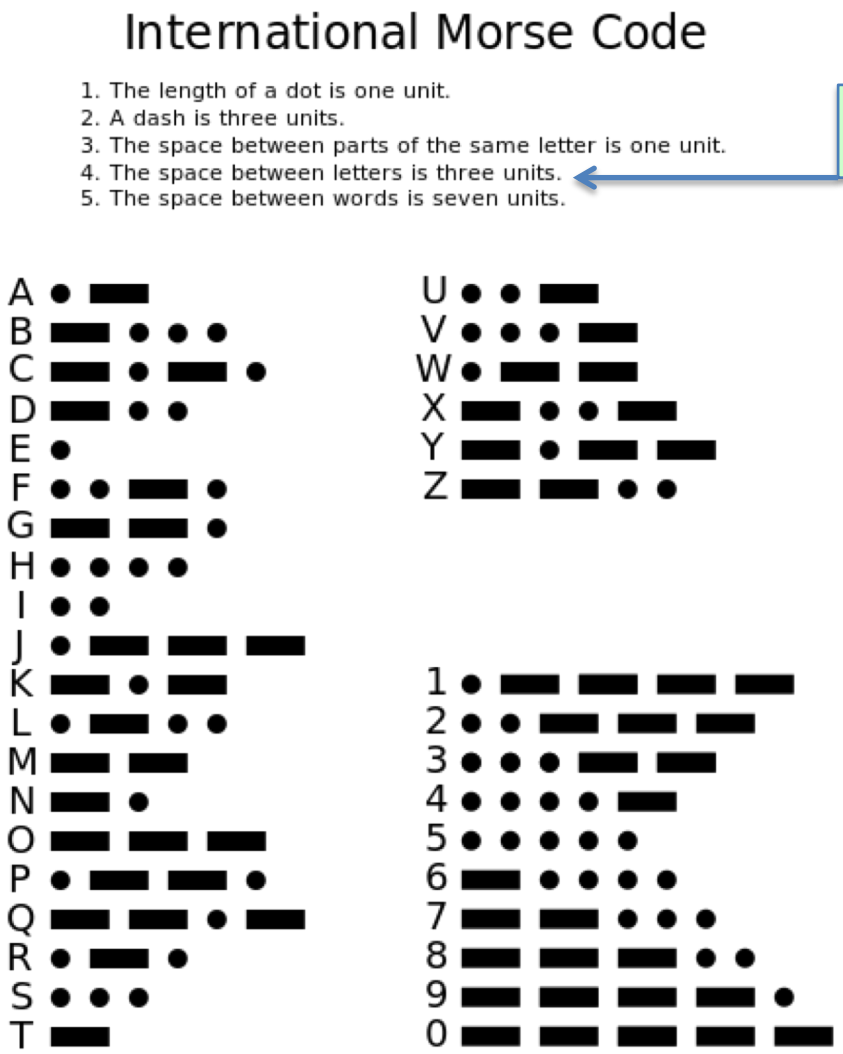
Weather	P	I[bit]	Codeword
Sun	0.5	1	00
Clouds	0.25	2	01
Rain	0.125	3	10
Snow	0.125	3	11
	$\Sigma\ p_i = 1.0$	$H_{src} = 1.75$	$S_{src} = 2$

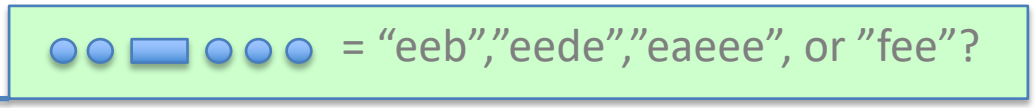
A fixed length code, always uses 2 bits per symbol

Entropy H_{src} = 1.75 bits/symbol, average codeword length S_{src} = 2 bits/symbol
Redundancy: ΔR = 0.25 bits/symbol

Obvious questions:
1. Can we do better?
2. If yes, then by how much exactly better?

Variable-length prefix codes: Morse code [1843]





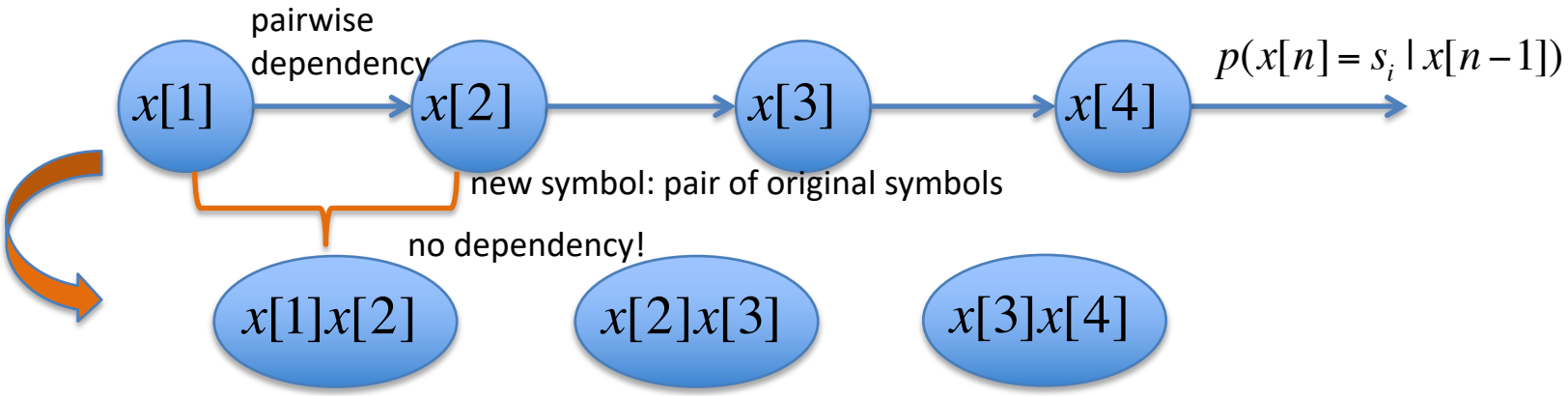
- Must necessarily be uniquely decodable!
- Ternary code: Z = {dot, dash, “3-unit pause”}
- Binary code: Z = {“low”, “high”}, and “3-unit pause” as obligatory suffix to each codeword
- Letter probabilities are different, so are the codeword lengths: spares time!
- First-order process model: most common letters (E, T) receive shorter codes, least common (J, Q, X, Z) – longer codes.
- Rudimentary second-order elements: e.g. a “space” cannot follow another “space”

Prefix-free (=prefix, =instantaneous) codes: no codeword is a prefix for another codeword

Later we will study codes which very efficiently solve the entropy coding problem!

Reducing an M-th order problem to a 1-st order problem

Example: 2-nd order Markov chain reduced to a 1-st order problem:



$$p(x[n]x[n-1] = s_i s_j) = p(x[n] = s_i \mid x[n-1] = s_j)\, p(x[n-1] = s_j),$$

having received x[n-1]: $p(x[n-1] = s_j) = 1$

However, this is not always practical:

- Alphabet size grows exponentially with *n*-gram length (i.e. as K^M)
- Higher order conditional probabilities become less accurate
  (need more training data to determine the probabilities of all new symbols)
- Not all data types are well-represented by finite (or lower-) order Markov processes:

> "three Swedish witches watch three Swiss Swatch watch switches.
> Which Swedish witch watches which Swiss Swatch watch switch?"

- very repetitive (i.e. compressible), but typical inter-letter correlation length is larger than e.g. 2

- **Instead:** perform data-matched de-correlation (transform), then apply entropy coding

General models: guessing inter-symbol dependencies

- Consider general (∞-order) model with complete dependencies on previous values:

$$p_i^{(n)} = p(x[n] = s_i \mid x[n-1], \ldots, x[0])$$

- Need to consider probabilities of all finite (sub)strings: $p(x[n], x[n-1], \ldots, x[0])$
- If we allow dependences between symbols, we might define "conditional entropy":
  (precise definition will be given later) $H_{cond} \le H_{src}$
- Define inter-symbol redundancy:
  (how much do we win by considering inter-symbol correlations?)

$$\Delta R_{cond} = H_{src} - H_{cond}$$

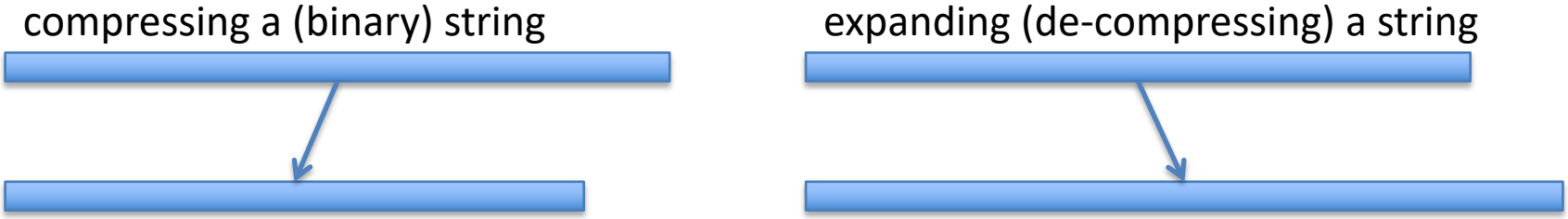
- **Modeling**: reduce (original) inter-symbol redundancy (after coding), optimally to zero
  - We cannot find the optimal solution (i.e. there exists no universal compression)
  - In general, an AI problem (and domain-dependent art)
  - There exists universal but uncomputable model (probability distribution) of data.

Example: let us code the digits of π = "3.14159265358979323846264338327950288419716…"

- As 0-th order model: 10 independent digits, $p_0 = p_1 = \ldots = p_9 = 0.1,$
  Shannon bound: $H_{src} \approx 3.3219$ [bits/digit], need > 415241 bytes for first million digits
- As a string: *"the first million digits of pi"* = 30 bytes, 0.00024 [bits/digit]!

Alternatively: write a program to compute π to any arbitrary length: up to 52 bytes in size!

General models: universal compression does not exist



- **Counting argument**: there are 2^n strings of length n, but only $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ strings shorter than n; ➔ no bijective mapping that reduces all strings.
- Fraction of strings of length n that can be compressed to m bits: at most $2^{(n-m)}$
- **Example**: fraction of strings (of any length) compressible by 1 byte is $\mathbf{2^{-8} = 0.4\%}$
- Every compressor *must* expand some inputs. However, expansion never needs to add more than one bit (e.g. return the unchanged input string + 1-bit indicator flag).
- **Recursive compression is prohibited**: optimally compressed data appears random to any algorithm, cannot be compressed further (but sub-optimally compressed data may be).

A Very Remarkable and Non-trivial Experimental Fact of Nature:
- most strings we care about (text, images, movies, software, DNA, …) are compressible!

Optimal modeling is not computable

[Solomonoff 1960, 1964], [Kolmogorov 1965], [Chaitin 1966]:

Algorithmic Probability: a universal (!) a priori (!) probability distribution over any strings

- Given a string x, compute following the sum over all programs M in language L that output x, with summand being $2^{-|M|}$, where $|M|$ is length of program M:

$$m(x) := \sum_{M:U(M)=x} 2^{-|M|}$$

Kolmogorov complexity of string x: $K_L(x) = |M|_{min}$; $m(x)$ above is dominated by $2^{-K(x)}$

- Invariance theorem: $K_L(x)$ in languages L_1 and L_2 differ only by an x-independent constant
  (Proof: given description M_1 in language L_1, output string x and append compiler from L_1 to L_2)

Best compression we can achieve for any string *x* is encoding it as the shortest program *M*

- Incomputability of $|M|_{min}$: assume an algorithm to find $|M|_{min}$ exists, enumerate its outputs; then *"the first string that cannot be described in million bits"* leads to a paradox.

Some minor remarks:

- It is not yet proven that the algorithmic probability is the true universal prior probability, but it is a widely accepted statement on empirical grounds
- **Minimal Description Length (MDL)** principle in Machine Learning: choose the simplest hypothesis that is consistent with the data
- **Occam's Razor** (14th century), basis of science: "choose the simplest answer that explains all facts"

Universal randomness test does not exist (again, no proofs)

[Gödel 1931]: first incompleteness theorem:
In any consistent formal system (set of axioms and rules of inference) powerful enough to describe rules of arithmetic, there is at least one true statement that cannot be proven.

[Li, Vitanyi 2007]: stronger variation of the Gödel's theorem
If the system is sound (cannot prove false statements), then there is an infinite number of true but un-provable statements. In particular, there is only a finite number of provable statements of type "x is random" ($K_L(x) \geq |x|$) and an infinite number of finite strings x.

Proof: suppose otherwise. Then one can enumerate all proofs (lists of axioms and applications of inference rules) and describe the paradoxical string

S = *"string x such that it is the first string to be proven to be a million bits long and random"*.

Similarly, there is no test to determine whether a compressed string can be compressed further (since this is equivalent to testing the compressed string for randomness).

Moral to take home: optimal compression is uncomputable.

A relatively “hard” example of a non-random string

Recipe to cook a non-random string:

- Take a string of million zero bytes (very non-random!)
- Encrypt it with the AES cryptographic algorithm in CBC mode with the key “foo”
- The actual result (a few first bytes, 1000032 bytes in total):

“

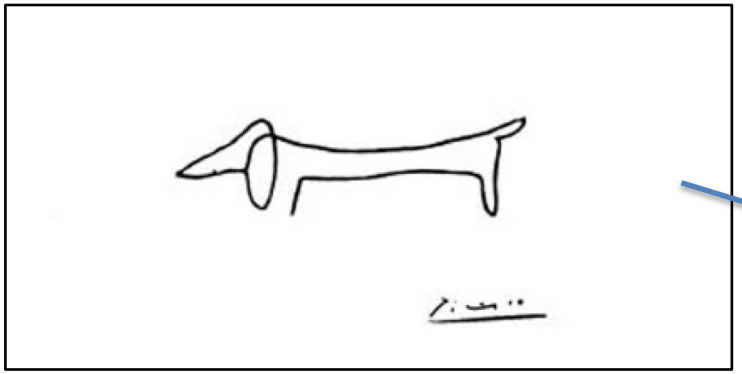
53 61 6C 74 65 64 5F 5F F0 A4 5F 70 56 1F 61 9B
9B 64 53 37 EF C6 22 D3 73 75 9E 46 F9 10 74 A6
9C 9A D5 08 F7 D8 33 A3 B7 91 9C D0 C3 CE D8 ED
39 F8 1A 63 F3 23 CB CF 5D 8C CC 14 8A 32 0C 58
5B 53 55 0A 6A 15 AC 84 8E EE D0 71 9D 9C A7 DD
...

“

- Looks (at least to me) pretty random (due to the design of the cryptographic algorithm), and incompressible almost to any algorithm [unless you know the key]!

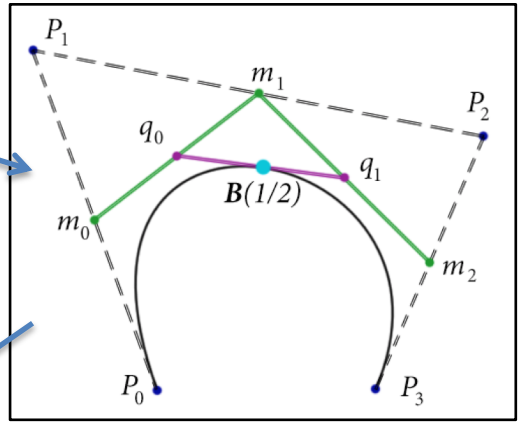
Example from image compression [Kun 2013]

"Dog" sketch by Picasso (costs many $M):

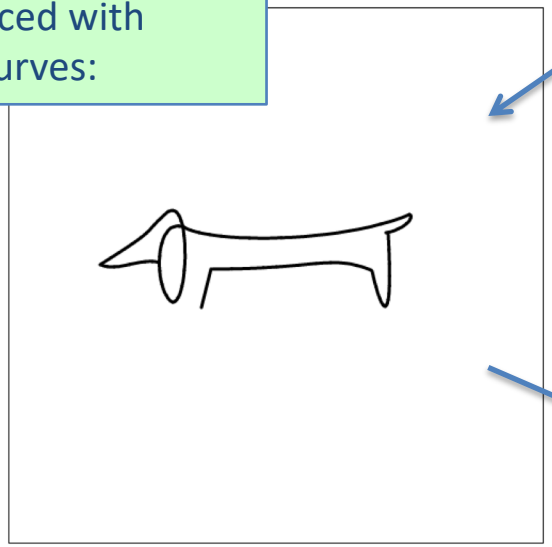


File in some graphics format, a few MB

Bezier polynomial representation of smooth curves (using a few key points and parameters):



The same drawing reproduced with Bezier curves:



Parameters of the resulting Bezier polynomials:

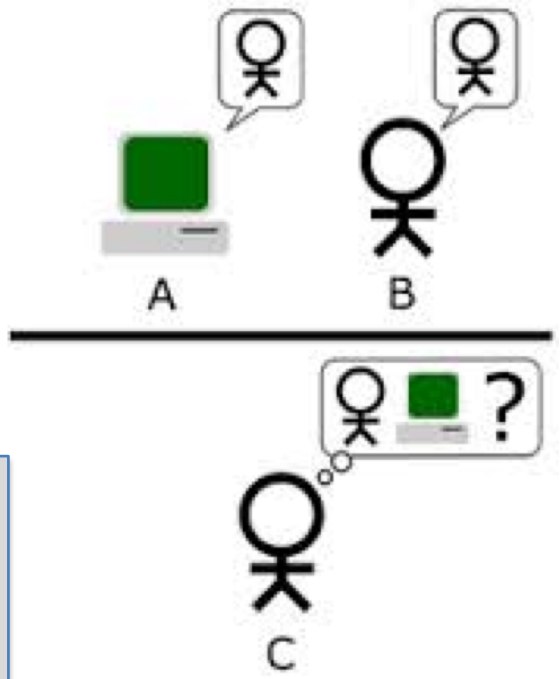
```
500 500
[180,280] [183,268] [186,256] [189,244] // front leg
[191,244] [290,244] [300,230] [339,245] // tummy
[340,246] [350,290] [360,300] [355,210] // back leg
[353,210] [370,207] [380,196] [375,193] // tail
[375,193] [310,220] [190,220] [164,205] // back
[164,205] [135,194] [135,265] [153,275] // ear start
[153,275] [168,275] [170,180] [150,190] // ear end + head
[149,190] [122,214] [142,204] [85,240]  // nose bridge
[86,240] [100,247] [125,233] [140,238]  // mouth
```

(Almost) equivalent representation, a few bytes!

Modeling of data as an Artificial Intelligence problem

Turing test from 50's on identifying the human intelligence:

- A human judge, human confederate, machine, and terminal.
- The confederate and the machine try to convince the judge that each of them is a human.
- If the judge cannot guess who is a machine correctly in 70% of time after 10 minutes of interaction, then the machine is declared to possess the Artificial Human Intelligence.



A possible transcript of an interaction session:

Q: Please write me a sonnet on the subject of the Forth Bridge.
A: Count me out on this one. I never could write poetry.
Q: Add 34957 to 70764.
A: (Pause about 30 seconds and then give as answer) 105621.
Q: Do you play chess?
A: Yes.
Q: I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?
A: (After a pause of 15 seconds) R-R8 mate.

To fool the judge, the machine needs to predict the human's answer to any given question.

Modeling of data as an Artificial Intelligence problem

- Prediction is intuitively related to understanding:
- *"Sic transit _____ mundi"* – fill in the blanks (if you can)

- [Legg, Hutter 2006]: optimal compression, if it were computable, would optimally solve AI problem under Turing test (1950) and the (more general) universal intelligence test.

Q: Please write me a sonnet on the subject of the Forth Bridge.
A: Count me out on this one. I never could write poetry.
Q: Add 34957 to 70764.
A: (Pause about 30 seconds and then give as answer) 105621.
Q: Do you play chess?
A: Yes.
Q: I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?
A: (After a pause of 15 seconds) R-R8 mate.

To compress the transcript above, we need a model:

response

context up to the current question

$$p(A \mid Q) = \frac{p(QA)}{p(Q)}$$

Can you guess, what could possibly be hidden behind this bar?

But: exactly the same model is needed to generate responses indistinguishable from those of a human, i.e. pass the Turing test!

Coding problem – summary (I)

- **Coding problem – general framework**
  - We consider a source that emits symbols $x[n] = s_i, S = \{s_0, \dots, s_{K-1}\}$
  - Coding: translate source symbols to a stream of bits (binary output alphabet)
  - Goal: reduce the redundancy (i.e. translation must be reversible and compact)
  - Stochastic approach: source is fully characterized by a stationary probabilistic model

$$p_i^{(n)} \equiv p(x[n] = s_i | x[n-1], x[n-2], \dots, x[0]).$$

- **Shannon [1948]: classification of models by the Markov process order**
- **1-st order processes (the most practically important case): $p_i^{(n)} = p_i$**
  - A very strong requirement on source statistics: no inter-symbol dependencies
  - In other words: $x[n]$ is independent of $x[m]$ for all $n \neq m$.
  - For a 1-st order model, we may define source entropy: $H_{src} = -\Sigma\, p_i \log_2(p_i)$
  - Codeword-based coding: given s_i, output some fixed binary string (a codeword)
  - For any coding scheme, average codeword length is $S_{src} = \Sigma\, p_i\, l_i$, where l_i is the length of a codeword (in bits) corresponding to the input symbol s_i.
  - Shannon boundary: $S_{src} \geq H_{src}$ for any reversible coding scheme.
  - Entropy coding schemes: coding under the 1-st order process assumption
  - There exist very efficient and universal algorithms for entropy coding

Coding problem – summary (II)

- **2-nd order processes:** $\boldsymbol{p_i^{(n)} = p(x[n] = s_i | x[n-1]) = \frac{p(x[n]=s_i, x[n-1]=s_j)}{p(x[n-1]=s_j)} = \frac{p_{ij}}{p_j}}$
  - A slightly more general model, slightly weaker independence assumptions
  - Any 2-nd order process can be represented as a 1-st order process!
  - Cost: growth of the alphabet size (digrams; K^2 symbols instead of K)

- **3-rd order, 4-th order processes, etc.**
  - Can be reduced to a 1-st order model by the same trick
  - Exponential alphabet size growth (i.e. need more data to determine probabilities)
  - Many important data types correspond to models of a very high finite order!
  - In practice, use many ad-hoc (heuristic) algorithms fitted to specific data types
  - "Pre-coding" step (reduces inter-symbol dependencies), then entropy coding

- **General coding problem: an ∞-th order process**
  - For simplicity, consider binary input (i.e. code binary strings -> binary strings)
  - A general model is equivalent to assigning probabilities to all finite binary strings:

$$p_i^{(n)} \equiv p(x[n] = s_i | x[n-1], x[n-2], \dots, x[0]) = \frac{p(x[n], x[n-1], \dots, x[0])}{p(x[n-1], \dots, x[0])}$$

Coding problem – summary (III)

- **General coding problem: ∞-th order process (continued)**
  - Coding: mapping a binary string x to a binary string y. Compression: $|y| < |x|$.
  - Counting argument: universal compression is impossible (some inputs are extended)

- **General coding problem: what is the best compression for a given string x?**
  - Amazingly, there exists a universal a priori PDF $p(x) = p(x[n], \dots, x[0])$
  - Consider all programs M in a language L that output x (i.e. all $M: U(M) = x$)
  - Algorithmic probability of a string x: $p(x) = \sum_{M:U(M)=x} 2^{-|M|}$
  - $p(x)$ is dominated by term 2^{-K_L}, where $K_L(x) = |M|_{min}$ is Kolmogorov complexity (length of the shortest program that outputs x)
  - Shortest program $M: U(M) = x$ is the best compression for string x!
  - Unfortunately, $K_L(x)$ is incomputable (no algorithm to compute $K_L(x)$ may exist)

- **General coding problem: is a given string x compressible?**
  - A string x is random if $K_L(x) \geq |x|$ (i.e. it is the shortest representation of itself)
  - A random string cannot be compressed further (i.e. it is incompressible)
  - Li, Vitanyi, 2007: no algorithm to check if x is random may exist

- **Therefore: optimal compression is impossible!** Only ad hoc data-fitted solutions...
  - Moreover: finding optimal compression = finding universal artificial intelligence