Neural Network Interest Group

Título/*Title*:

Data Compression Using NNs

Autor(es)/*Author(s)*:
Luís Alexandre

Relatório Técnico/*Technical Report* No. 10 /2006

FEUP/INEB, Rua Dr. Roberto Frias, 4200-465, Porto, Portugal

Título/*Title*:

Data Compression Using NNs

Autor(es)/*Author(s)*:

Luís Alexandre

Relatório Técnico/*Technical Report* No. 10 /2006

Publicado por/*Published by*: NNIG. http://paginas.fe.up.pt/~nnig/

Data compression using NNs

Luís A. Alexandre

October 27, 2006

Introduction

- Any type of data can be viewed as a string of 0s and 1s (bits)
- Compression is the act of converting a original string of m bits into another of $n < m$ bits.
- Of course we assume that there is another algorithm called a decompressor that can do the inverse of the compressor and produce the original file from the compressed file.
- Compression can be either lossless or lossy. I will talk about lossless compression.

There is no universal compressor

- Imagine that a universal compressor existed: any string of size m could be compressed into another of size $n < m$.
- There is no such algorithm.
- Proof 1: If a u.c. existed then it could be applied to any string recursively until its size would be 0.
- Proof 2: Consider an original string of size m. It would be compressed into one of size $n < m$. There are $\sum_{i=0}^{m-1} 2^i = 2^{m-1}$ such strings. Since we could apply the compressor to all 2^m strings of size m, it means that at least 2 strings of size m would have to map to the same string of size $n < m$. Of course decompression into the original strings would now be impossible.

Implications

- The fact that no u.c. exists means that the compressors will work better on some type of data than on others.
- This opens the way for the production of data specific compression algorithms: JPEG for images, mp3 for sound, etc.
- I will be concerned with text compression. Although for now it wont affect the presentation.

Families of compression algorithms

- Lempel-Ziv (zip, gzip): there is no separation between modelling and coding. The compression consists in replacing a substring with a pointer to an earlier occurrence of the same substring.
- Burrows-Wheeler (bzip2, szip): The transformation rearranges the order of the characters. If the original string had several substrings that occurred often, then the transformed string will have several places where a single character is repeated multiple times in a row. This is useful for compression, since it tends to be easy to compress a string that has runs of repeated characters.
- PPM: see next section.

Dynamic coding

- Usually the compressors explore the non-uniform distribution of text sequences: it is much more probable to find the sequence "the"than the sequence "zqr". So the first is coded with a smaller codeword.
- But more can be done than this: there are context dependencies that can be explored
- Example: if the word 'fire' occurs than its is much more probable to have nearby the word 'fireman' that the word "flower". So the coding scheme should be dynamic and context aware.
- This is were PPM enters.

PPM

- PPM is an adaptive statistical data compression technique based on context modeling and prediction.
- PPM models use a set of previous symbols in the uncompressed symbol stream to predict the next symbol in the stream.

Compression = prediction + coding

- For PPM, compression has two parts: there is a prediction (and modeling) part and a coding part.
- We will start by talking about the coding part in the next slides.

Entropy encoding

- An entropy encoding is a coding scheme that assigns codes to symbols so as to match code lengths with the probabilities of the symbols.
- Typically, entropy encoders are used to compress data by replacing symbols represented by equal-length codes with symbols represented by codes where the length of each codeword is proportional to the negative logarithm of the probability. Therefore, the most common symbols use the shortest codes.

Entropy encoding: example

Message: caaabb

prob. of each symbol: a=1/2, b=1/3, c=1/6

Symbol	equal length code	entropic code
a	00	0
b	01	10
c	11	110
coded string	110000000101	1100001010
BPC	2.0	1.(6)

Huffman coding

- This is the best symbol code algorithm: the one that produces codes with expected length closer to the optimal that is the entropy of the string.
- For the English language symbols it gives an expected length of 4.15 bits. The entropy is 4.11 bits.

x	$P(x)$
a	0.0575
b	0.0128
c	0.0263
d	0.0285
e	0.0913
f	0.0173
g	0.0133
h	0.0313
i	0.0599
j	0.0006
k	0.0084
l	0.0335
m	0.0235
n	0.0596
o	0.0689
p	0.0192
q	0.0008
r	0.0508
s	0.0567
t	0.0706
u	0.0334
v	0.0069
w	0.0119
x	0.0073
y	0.0164
z	0.0007
–	0.1928

(from [1])

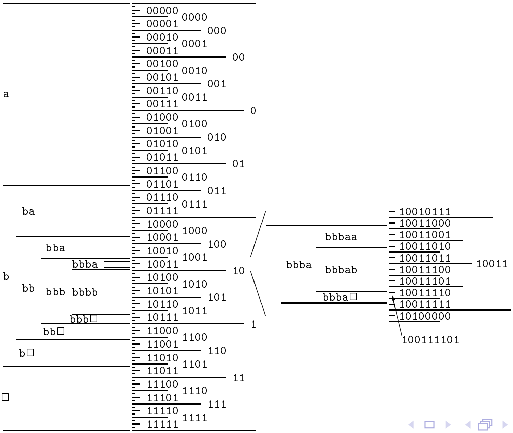
Arithmetic coding

- Typically, entropy encoding methods separate the input message into its component symbols and replace each symbol with a code word.
- Arithmetic coding encodes the entire message into a single number $n \in [0, 1)$.
- This way, arithmetic coding can produce a smaller code for a block of text than the codes produced by Huffman coding.

Arithmetic coding: example (from [1])

- In this example we want to code the sequence of results that comes from a toss of a coin: a, b. We will also use the symbol $\Box$ to denote the end of the experiment.
- Let's consider the string to be $bbba\Box$
- We pass along the string one symbol at a time and use our model to find the probability distribution of the next symbol given the string so far.
- Larger intervals require fewer bits to encode.

Arithmetic coding: example (from [1])



NNs for prediction

- The idea here is to adapt PPM and use NNs for the prediction stage.
- The problem with tradicional NNs is that training takes a long time and is thus not adequate for compression applications: can be as much as 1000 times slower than other compressors.
- Mahoney [2] introduced a fast NN adapted to text compression.

Mahoney's fast NNs

- We need to find a model for the $P(y|\mathbf{x})$, where $y \in \{0,1\}$ and $\mathbf{x}$ is a context vector where each component x_i is a context (the last bit, the last n bits, etc.)
- It happens that the most likely distribution is given by $P(y|\mathbf{x}) = \frac{1}{Z}\exp(\sum_i w_i x_i)$
- If we choose Z such that $P(y=0|\mathbf{x}) + P(y=1|\mathbf{x}) = 1$ then $P(y|\mathbf{x}) = g(\sum_i w_i x_i)$ where $g(x) = 1/(1+\exp(-x))$
- This represents a perceptron where the x_i are the inputs and the w_i are the weights. The output represents $P(y=1|\mathbf{x})$.

Example results for text compression

- The following results (in bpc) are from [2]
- The last line was obtained using the PAQ8h compressor from `http://cs.fit.edu/~mmahoney/compression/`

Program	Year	Type	Alice	Book1
gzip	1993	LZ	2.848	3.250
szip	1998	BW	2.239	2.345
rkive	1998	PPM	2.055	2.120
P6	1999	NN	2.129	2.283
PAQ8h	2006	NN	1.816	2.001

David J.C. MacKay.
Information Theory, Inference and Learning Algorithms.
Cambridge University Press, 2003.

M.V. Mahoney.
Fast text compression with neural networks.
In *Proceedings of the FLAIRS 2000*, Florida, USA, May 2000.