Arabic Text Compression
Multimedia Course Project

Ahmad Essam	Ahmad Khaled	Ibrahim Mohammed	Omnia Zakaria
Section 1, BN. 7	*Section 1, BN. 3*	*Section 1, BN. 1*	*Section 1, BN. 13*

Abstract

We compress a database of twenty Arabic text files consisting of reviews scrapped from the web. We present a brief recounting of the compression algorithms we've tried and an overview of the algorithm with the best performance (PAQ).

Contents

Table 1. History of Experiments

Algorithm	Compression Ratio	Implementer
Huffman Coding	3.13	I. Mahmoud
LZ77	∼3	A. Essam
LZ77 + Huffman	4.4	A. Essam
Arithmetic Coding with Zero Order Context	∼2	A. Khaled
Arithmetic Coding with Linear Context Mixing	∼2	A. Khaled
LZW	2	O. Zakaria
PPMA	5.1	A. Essam
PPM with Context Blending	5.4	A. Essam
PAQ7 variant	5.49	A. Khaled

1. Introduction

Modern statistical methods have enjoyed remarkable success in recent years in various fields. Data compression is an area which statistical methods have come to dominate in the recent years: all the recent winning submissions to the Hutter Prize (the most important prize in the field) have been programs which heavily utilized statistical algorithms. In general, when using methods such as arithmetic encoding or Huffman coding, the compression algorithm is as good as the probability model of the data that it has, hence statistical models enable us to reach higher compression rates by inferring more information about the data source from the file being compressed.

2. History of Experiments

In Table 1 we present the algorithms we implemented in previous attempts alongside the achieved compression rate for each.

3. The PAQ Algorithm

The PAQ algorithm is a well-known (in the data compression community) compression algorithm with state-of-the-art results that compresses files bit by bit using integer arithmetic encoding, It relies on modelling the data in a file by a large number of probability models, mixing the predictions of the model using a neural network, continuously improving the neural network's ability to predict the next bit given the previous contexts. We give a short overview of Neural Networks next, then discuss the specific implementation in PAQ.

3.1. Neural Networks

Neural Networks are a specific class of learning methods whose central idea is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features. [2, p.389]
We restrict ourselves to two-layer neural networks with a single hidden layer and with sigmoidal activation functions, in this case neural networks are N-dimensional vector functions where the k-th element is of the following form:

$$f(x)_k = g\left(\sum_{i=1}^{L} \alpha_{ki}\sigma\left(\sum_{j=1}^{D} w_i x_i + b_1\right) + b_2\right)$$

where g is an *output function*, σ is the sigmoid function defined by $\sigma(x) = \frac{1}{1+e^{-x}}$, and $\alpha_{ki}, w_i, b_i \in \mathbb{R}$. A neural network is usually represented as being comprised of multiple layers, and for the neural networks considered here there is only an input layer (containing the x_i), a single hidden layer (defined by $h_i = \sigma\left(\sum_{j=1}^{D} w_i x_i + b_1\right)$) and an output layer (defined by $y_k = g\left(\sum_{i=1}^{L} \alpha_{ki} h_i + b_2\right)$). It is also customary to define $a_i = \sum_{j=1}^{D} w_i x_i + b_1$ to be the *pre-activation* of the i-th hidden layer.

The equations of *forward propagation* (computing the output of the neural network given certain inputs and weights) are

$$a_i = \sum_{j=1}^{D} w_i x_i + b_1 \text{ for all } i \text{ such that } 1 \leq i \leq L$$
$$h_i = \sigma(a_i) \text{ for all } i \text{ such that } 1 \leq i \leq L$$
$$y_k = g\left(\sum_{i=1}^{L} \alpha_{ki} h_i + b_2\right) \text{ for all } i \text{ such that } 1 \leq i \leq N$$

And these equations allow us to compute the outputs y_i of the neural network given the inputs x_i and the parameters θ. Since we are interested in *learning*, we want to find a way to change the parameters θ as a response to some loss or error function E (because unlike linear models the luxury of closed-form solutions is rarely, if ever, available with neural networks), and the way to do this is through *gradient descent*: we find the gradient of the error function with respect to each of the parameters and update the parameters as follows:

$$\theta = \theta - \alpha \nabla_\theta E(x, y, \theta)$$

where E is our error function calculated for training data (x, y) and parameters θ, and α is the *learning rate*, a coefficient which determines how much we are going to change the parameters by the gradient each iteration. This error update process may be done locally (as in PAQ) or nonlocally. We say that neural networks learn in "iterations" where they process an input-output pair (x_i, y_i) and update their parameters.
It is not immediately obvious why we should use neural networks for machine learning. Neural networks can approximate any function approximately well, under some mild assumptions on said function [1]. Furthermore, the use of neural networks with many layers has been remarkably successful in the last few on tasks which were previously very hard for computers. [4] In PAQ, the neural network has two layers with one hidden layer. In addition, there's another neural network that combines the outputs in the output layer to a single prediction.

3.2. Stochastic Gradient Descent

The technique described in the previous section does gradient descent using *all* the inputs every iteration, which is very unfeasible when the number of parameters is very large. An alternative technique is stochastic process: which instead approximates the gradient of the loss function at all the inputs with the gradient at a few inputs. Such an approximation enables realistic training times for the learning methods.

3.3. Cross Entropy Error

The traditional error metric used for propagating errors in neural networks is the least squares error, however, the error metric used in PAQ and similar algorithms is the cross entropy error which minimizes the predictive cost for the last bit encoded or decoded, which makes sense since we want to minimize the predictive error directly.

3.4. Mixture of Experts

PAQ uses a mixture of experts architecture in which most of the neural network is inactive at a single time and only a few experts are active. The experts are selected according to a deterministic weighting algorithm, refer to [3] for the algorithm.

3.5. Context Models

Nonstationary context maps predict the next bit given a number of bytes. Nonstationary context maps traditionally have a state represented by a pair of counts (n_0, n_1) where n_0 is the number of times a 0 was seen following this context and n_1 is the number of times a 1 was seen following this context. We mix a large number of context models. The prediction is then $p(1) = \frac{n_1}{n_0+n_1}$, however in order to correctly model non-stationary data (data in which the probabilities change over time) the counts n_0 and n_1 are mapped to probabilities using a statemap which is altered each step so as to minimize the prediction error. Run Context maps model continuous runs of bits given contexts and are useful for repetitive data. A context map outputs the predictions from a non-stationary context map and a run context map for a certain context length.

3.6. Difference from canonical PAQ

We did not implement a few models: the sparse model, the record model, the image and audio specific models, and we didn't implement the adaptive probability map which is used for improving the predictions of the neural network. In addition, we added a parser that converts Arabic and English UTF-8 characters to indices first before compressing it using PAQ.

4. Conclusion and Future Work

Due to time constraints we didn't implement many of PAQ's models or PAQ's Secondary Symbol Estimation (SSE or APM) system, and our implementation is also limited in speed by not using a vectorized neural network (we use plain for loops in C++). A future implementation would solve these problems and also explore the additions to PAQ mentioned in [3].

5. Content of Compressed Files

The compressed files contain a 28-bit header containing the size of the file encoded and then a tag of arbitrary length for arithmetic encoding. When the arithmetic encoded file is decoded by PAQ, it is then decoded by the UTF parser that returns the original file.

6. Workload Division

The work division is mentioned in Table 1.

7. References

[1] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[3] Byron Knoll and Nando de Freitas. "A Machine Learning Perspective on Predictive Coding with PAQ". In: *CoRR* abs/1108.3298 (2011). arXiv: 1108.3298. URL: http://arxiv.org/abs/1108.3298.

[4] H. W. Lin, M. Tegmark, and D. Rolnick. "Why Does Deep and Cheap Learning Work So Well?" In: *Journal of Statistical Physics* 168 (Sept. 2017), pp. 1223–1247. DOI: 10.1007/s10955-017-1836-5. arXiv: 1608.08225 [cond-mat.dis-nn].