Bolt: Accelerated Data Mining with Fast Vector Compression

Davis W. Blalock
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
dblalock@mit.edu

John V. Guttag
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
guttag@mit.edu

ABSTRACT

v1 (partial)

The need to store and operate on vectors of numbers is at the heart of data mining. However, doing so can require great amounts of storage space, memory, and computation time. A great deal of work has been done on generating compressed representations to reduce this cost, but most existing work either assumes data characteristics such as sparsity, preserves only aggregate statistics, or focuses on slow, one-time encoding of large databases.

v2

Traditionally, compression has been a tool only to reduce space usage, with some recent work on learning compressed representations to accelerate similarity search or neural network inference. We describe an algorithm that allows compression to be used to accelerate virtually any algorithm using vector distances or dot products. Specifically, by learning a compressed representation on which vector operations can be approximated directly, we can both reduce space consumption and increase speed by 10× or more with little or no loss in accuracy. Moreover, and in sharpest contrast to existing work, our technique can encode vectors at over 4GB/s in a single CPU thread, making conversion to our representation nearly free and worthwhile even on fast-changing data such as model parameters.

We show experimentally that our approach can be used to accelerate neural network training, k-means clustering, nearest neighbor search, and maximum inner product search by up to 20×. Furthermore, our approximate Euclidean distance and dot product computations are faster not only than those of related algorithms with much slower encodings, but even faster than Hamming distance computations, which have direct hardware support on the tested platforms.

CCS CONCEPTS

•**Mathematics of computing →Probability and statistics;** *Probabilistic algorithms; Dimensionality reduction;* Mathematical software;

ACM Reference format:
Davis W. Blalock and John V. Guttag. 2017. Bolt: Accelerated Data Mining with Fast Vector Compression. In *Proceedings of ACM SIGKDD, Halifax, Nova Scotia Canada, August 2017 (KDD 2017),* 2 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

KDD 2017, Halifax, Nova Scotia Canada

DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Many machine learning algorithms are built around some combination of matrix multiplications, dot products, and distance computations in vector spaces. Such algorithms include deep neural networks, k-means and hierarchical clustering, and linear regression, among many others.

Lots of people have tried speeding up deep neural nets, mostly by pruning them post-hoc or adopting elementwise quantization / sparsity

for ease of exposition, we will refer to computing "similarities" as shorthand for computing either dot products or euclidean distances throughout the remainder of this work, because typing "dot products or distances" is really verbose

Contributions:

A fast means of preprocessing vectors so as to minimize quantization loss. This method is compatible with most techniques in the rapidly-evolving area of vector quantization.

A hardware-friendly means of computing approximate distances using quantized vectors.

Theoretical analysis of our technique's effectiveness.

Empirical demonstration that approximate distances suffice for many real-world applications.

somewhere sneak in the fact that Bolt is an acronym for Based On Lookup Tables

problem statement

2 RELATED WORK

Great deal of work on vector quantization -mostly on generating better encoding algorithms -typically through complex procedures that require tons of encoding time -eg, state of the art (LSQ) encoding takes over a ms *per vector*; we're like a ten thousand times faster than this -sometimes through direct integration with an index, such as the IMI (LOPQ does this) -our scanning algorithm can use any of these encoding schemes that can spit out 4bit codes, and our preprocessing can help with basically any of them -most similar to ours is that VLDB paper that used the same machine instructions; however, it required huge sets of static vectors that could be sorted ahead of time

-also lots of work on compressing neural nets -replacing raw mats with structured mats, such as adaptive fastfood transform or the DCT ones -heuristic (?) weight-sharing like hashnets -zelda diversity nets -quantization, sparsity -pruning post-hoc -learning with only quantized reprs -ASIC acceleration -other stuff I don't know that well

-sketching / embedding -pretty sure there's some sketch for euclidean distance -sketching and embedding equivalent for norms

-sketching for stats such as number of uniques

3 METHOD

overview -as mentioned in the problem statement, our goal is to construct a distance (or similarity) function $\hat{d}(q, x)$ that approximates some true distance (or similarity) function d(q, x).

-Further recall that $\hat{d}$ need not operate on the original spaces Q and X, but can instead use transformed spaces $G = g(Q)$ and $H = h(X)$, so that its signature is instead $\hat{d} : GxH \rightarrow R$.

-our method entails offline learning of the functions G and H in a manner similar to existing Multi-Codebook Quantization (MCQ) techniques, and online computation of the distance

-we first describe the function $h(.)$, which quantizes the vectors X.

-we then describe the functions $g(.)$, which computes sufficient statistics about the vector q

-we finally describe the function $\hat{d}(.,.)$, which returns a distance based on $g(q)$ and $h(x)$.

Quantization scheme -product quantization background -our novel permutation method

Distance computation -asymetric distance computation background

4 RESULTS

show that it makes matrix-vector muls way faster for various matrix sizes -also show approximation error -which suggests getting lots of meaningful matrices, ideally fc layer weights -prolly have to show both speed and err as a function of code length same for matrix-matrix; speed and acc -is there any way we'll be better here? maybe for skinny mats...

show that we can speed up kmeans a lot, and point out that this is orthogonal to other ppl's speedups based on pruning which ones you consider moving and/or using minibatches -prolly pick a couple datasets and show it as a function of k

a couple experiments on sift1m scan and mnist scan, prolly; maybe sift10m / deep10m also

somewhere introduce the "cold scan" problem, wherein we also have to add in time to compress everything, but then we get a batch of queries -if just 1 query, you're always worse off doing the encoding -metric of interest is how many queries you have to do @k database vectors before it's faster than a matmul -and you should also report the times for different query counts and db counts

-and prolly also the "warm scan" problem; you only have to encode some subset of the db (cuz it changed)

-or maybe the "hot scan" problem cuz data is hot; in contrast to fixed data, which is cold/frozen

5 CONCLUSION