PARMAC: DISTRIBUTED OPTIMISATION OF NESTED FUNCTIONS, WITH APPLICATION TO LEARNING BINARY AUTOENCODERS

Miguel A. Carreira-Perpiñan and Mehdi Alizadeh, EECS, UC Merced

Abstract

Many powerful machine learning models are based on the composition of multiple processing layers, such as deep nets, which gives rise to nonconvex objective functions. A general, recent approach to optimise such “nested” functions is the method of auxiliary coordinates (MAC). MAC introduces an auxiliary coordinate for each data point in order to decouple the nested model into independent submodels. This decomposes the optimisation into steps that alternate between training single layers and updating the coordinates. It has the advantage that it reuses existing single-layer algorithms, introduces parallelism, and does not need to use chain-rule gradients, so it works with nondifferentiable layers.

We describe the case of the composition of two functions, in particular the quadratic penalty: minimize ∑₀≤i<n fᵢ(xᵢ), subject to h(xᵢ) = 0. The optimal solution X* corresponds to xᵢ = h⁻¹(0). We study the convergence of MAC and its parallel speedup, and implement PARMAC using MPI to learn binary autoencoders for fast image retrieval, achieving nearly perfect speedups in a 128-processor cluster, with a training set of 128 billion images.

1. ParMAC for binary autoencoders

MAC introduces embarrassing parallelism within each step: L = D independent submodels in the W step and V independent codes in the Z step. We now show how to turn this into a distributed, low-communication PARMAC algorithm. In general, assume separability over the \( Y \) data points in the Z step, and separability over the \( W \) submodels in the W step.

With large datasets in distributed systems, it is imperative to minimize data movement over the network because the communication time generally far exceeds the computation time in modern architectures. In MAC we have 3 types of data: the original training data \( \{X_i , Y_i\} \), the auxiliary coordinates \( Z \), and the submodel parameters (\( \theta \), the hash functions \( \mathbf{h} \) and decoder \( f \), for BAs).

In ParMAC, we never communicate between training single layers and updating the coordinates. Each machine keeps a disjoint portion of \( \{X_i , Y_i , Z\} \), corresponding to a subset of the points. Only model parameters are communicated, during the W step, following a circular topology which implicitly implements a stochastic optimisation.

Using \( P \) machines, ParMAC iterates as follows:

- **Z step**: identical to MAC, each point’s coordinates \( x_i \) are optimized independently, in parallel over machines (since each machine contains \( x_i, x_j \), and all the model parameters).
- **W step**: for the hash functions \( \mathbf{h}(w_i) \) and decoders \( f(L_i) \) visit each machine. This means they train with stochastic gradient descent: one “epoch” for model \( W \), corresponding to \( w_i \), having visited all \( P \) machines (within each machine, data are also split into minibatches).

2. How does the communication occur? In the Z step, there is no communication at all.

- With \( \epsilon \) epochs, the entire model parameters are communicated \( \epsilon + 1 \) times. The last round of communication is needed to ensure each machine has the most updated version of the global model before the next Z step.
- This communication can be implemented with a circular topology.
- This introduces no bottlenecks, unlike the use of parameter servers that gather parameters from the workers, update them, and broadcast them back to the workers.
- We can also run \( \epsilon \) epochs with only 2 rounds of communication by having a submodel do \( \epsilon \) consecutive passes within each machine’s data.

Theoretical model of the parallel speedup \( S_P \) with \( P \) machines: in a practical setting where communication dominates computation and the dataset is large, the speedup is nearly proportional to the number of machines. It peaks at \( S_P = S_R > M \) for \( P > P^* > M \) and decreases thereafter as \( P \to \infty \).

Convergence guarantees (to a local optimum): since the only approximation to the original MAC algorithm is incurred by using SGD in the W step, and we can guarantee convergence of SGD under certain conditions, we recover the original convergence guarantees for MAC.

3. Experiments on a distributed cluster

We have trained BAs (with linear and kernel hash functions, in image datasets for information retrieval) with ParMAC, implemented in C with MPI, in a distributed cluster (UCSD Triton Shared Computer Cluster, with 128 nodes having 4 GB RAM per core).

- The theoretical speedup matches well the experimental one.
- The speedups we achieve are on average perfect for \( P < M \) and hold very well for \( P > M \) up to the maximum number of machines we used (\( P < 128 \)).

Runtime in the SIFT-1B dataset using \( L = 64 \) bits, \( \epsilon = 2 \) epochs and \( P = 128 \) machines: 29 hours (linear SVM) and 83 hours (kernel SVM). This would take months in a single machine with enough RAM to hold the data and parameters (which would require several TB).

### Speedup \( S_P \) as a function of the number of machines \( P \):

- CIFAR
  - \( N = 58K, M = 32 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).
- SIFT-1M
  - \( N = 10^7, M = 32 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).
- SIFT-1B
  - \( N = 10^9, M = 128 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).

### Model

- \( X \) : training data
- \( Y \) : output
- \( Z \) : auxiliary coordinates
- \( W \) : submodels
- \( P \) : number of machines
- \( V \) : number of submodels
- \( \epsilon \) : number of training epochs
- \( \mathbf{h} \) : hash functions
- \( f \) : decoders

### Data

- CIFAR
  - \( N = 58K, M = 32 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).
- SIFT-1M
  - \( N = 10^7, M = 32 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).
- SIFT-1B
  - \( N = 10^9, M = 128 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).

### Number of Machines

- CIFAR
  - \( N = 58K, M = 32 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).
- SIFT-1M
  - \( N = 10^7, M = 32 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).
- SIFT-1B
  - \( N = 10^9, M = 128 \)
  - \( \epsilon = 3 \) epochs
  - \( S_P \) increased from \( P = 1 \) to \( P = 128 \), and held very well for \( P > 128 \).