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

Miguel Á. Carreira-Perpiñán and **Mehdi Alizadeh**, EECS, UC Merced

1 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 ParMAC, a distributed-computation model for MAC. This trains on a dataset distributed across machines while limiting the amount of communication so it does not obliterate the benefit of parallelism. ParMAC works on a cluster of machines with a circular topology and alternates two steps until convergence: one step trains the submodels in parallel using stochastic updates, and the other trains the coordinates in parallel. Only submodel parameters, no data or coordinates, are ever communicated between machines. ParMAC exhibits high parallelism, low communication overhead, and facilitates data shuffling, load balancing, fault tolerance and streaming data processing. We study the convergence of ParMAC 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 100M images.

Supported by NSF awards CAREER IIS–0754089, IIS–1423515 and a Google Faculty Research Award.

2 MAC for binary autoencoders

MAC and ParMAC apply generally to nested functions with multiple layers. We describe the case of the composition of two functions, in particular the binary autoencoder (BA) (Carreira-Perpiñán & Raziperchikolaei, 2015): given a dataset $\mathbf{X}_{D\times N} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$, we minimize:

$$E_{\text{BA}}(\mathbf{h}, \mathbf{f}) = \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{f}(\mathbf{h}(\mathbf{x}_n))\|^2 \quad \begin{cases} \text{encoder } \mathbf{h}\text{: } \mathbb{R}^D \to \{0,1\}^L \\ \text{decoder } \mathbf{f}\text{: } \{0,1\}^L \to \mathbb{R}^D. \end{cases}$$

A BA is like a usual autoencoder but with a binary code layer. It defines a nonsmooth, nonconvex problem; chain-rule gradients are inapplicable.

The MAC design pattern: ❶ introduce an auxiliary coordinate $\mathbf{z}_n = \mathbf{h}(\mathbf{x}_n) \in \{0,1\}^L$ for each data point, corresponding to its L-bit code vector (in the bottleneck layer), in order to decouple the nesting. ❷ Apply a penalty method to the constrained problem, e.g. the quadratic penalty: minimize, while driving $\mu \to \infty$:

$$E_Q(\mathbf{h}, \mathbf{f}, \mathbf{Z}; \mu) = \sum_{n=1}^{N} \left(\|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 \right) \quad \text{s.t.} \quad \mathbf{z}_1, \dots, \mathbf{z}_N \in \{0,1\}^L$$

❸ Alternating optimisation of E_Q over coordinates $\mathbf{Z}$ and submodels $\mathbf{W} = (\mathbf{h}, \mathbf{f})$:

- Coordinates: over $\mathbf{Z}$ for fixed $(\mathbf{h}, \mathbf{f})$, it separates into N independent optimisations over the N codes $\mathbf{z}_1, \dots, \mathbf{z}_N$, each on only L bits. Solved by enumeration if L is small or approximately by alternating optimisation over bits.
- Submodels: over $\mathbf{W} = (\mathbf{h}, \mathbf{f})$ for fixed $\mathbf{Z}$, we obtain $L+D$ independent problems (L single-bit hash functions, D decoders). With linear $\mathbf{h}$, $\mathbf{f}$ this simply involves fitting L SVMs to $(\mathbf{X}, \mathbf{Z})$ and D linear regressors to $(\mathbf{Z}, \mathbf{X})$.

Training BAs with MAC beats approximate approaches such as relaxing the codes or the step function in the encoder. It yields state-of-the-art binary hash functions $\mathbf{h}$, which can be used for fast approximate nearest-neighbor search using Hamming distances (e.g. for image retrieval).

3 ParMAC for binary autoencoders

MAC introduces embarrasing parallelism within each step: $L + D$ independent submodels in the $\mathbf{W}$ step and N independent codes in the $\mathbf{Z}$ step. We now show how to turn this into a distributed, low-communication **ParMAC** algorithm. In general, assume separability over the N data points in the $\mathbf{Z}$ step, and separability over the M submodels in the $\mathbf{W}$ 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 $(\mathbf{X}, \mathbf{Y})$, the auxiliary coordinates $\mathbf{Z}$, and the submodel parameters (= the hash functions $\mathbf{h}$ and decoder $\mathbf{f}$, for BAs). Usually, the latter type is far smaller. In ParMAC, we never communicate training or coordinate data; each machine keeps a disjoint portion of $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ corresponding to a subset of the points. Only model parameters are communicated, during the $\mathbf{W}$ step, following a circular topology which implicitly implements a stochastic optimisation.

Using P machines, ParMAC iterates as follows:

- $\mathbf{Z}$ step: identical to MAC, each point's coordinates $\mathbf{z}_n$ are optimized independently, in parallel over machines (since each machine contains $\mathbf{x}_n$, $\mathbf{z}_n$, and all the model parameters).
- $\mathbf{W}$ step: the hash functions $\{\mathbf{w}_h\}$ and decoders $\{\mathbf{f}_d\}$ visit each machine. This implies we train them with stochastic gradient descent: one "epoch" for model $\mathbf{w}_h$ corresponds to $\mathbf{w}_h$ having visited all P machines (within each machine, data are also split into minibatches).

How does the communication occur? In the $\mathbf{Z}$ step, there is no communication at all. In the $\mathbf{W}$ step, all M submodels $\{\mathbf{w}_h\}$ are communicated asynchronously in parallel:

- With e epochs, the entire model parameters are communicated $e+1$ times. The last round of communication is needed to ensure each machine has the most updated version of the entire model for the $\mathbf{Z}$ 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 e epochs with only 2 rounds of communication by having a submodel do e consecutive passes within each machine's data.

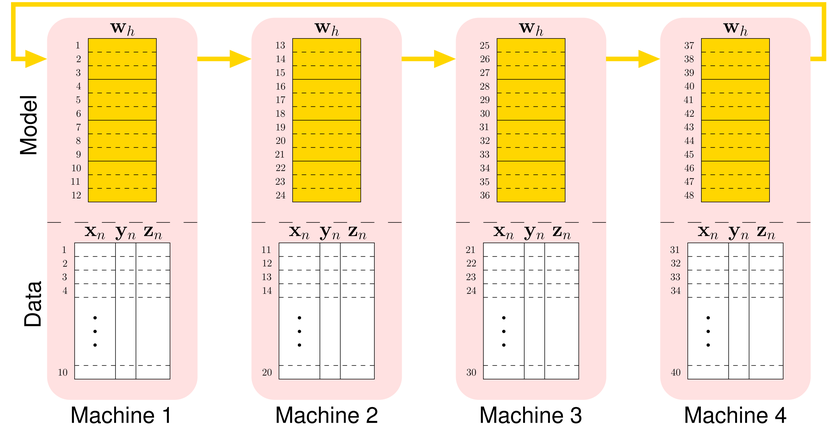
Theoretical model of the parallel speedup $S(P)$ with P machines: in a practical setting where communication dominates computation and the dataset is large, the speedup is nearly perfect ($S(P) \approx P$) if using fewer machines than submodels ($P \leq M$), and otherwise it peaks at $S(P) = S_1^* > M$ for $P = P_1^* > M$ 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 $\mathbf{W}$ step, and we can guarantee convergence of SGD under certain conditions, we recover the original convergence guarantees for MAC.

Extensions of ParMAC:

- Data shuffling: access local data in random order (within-machine) or randomise the circular topology at each epoch (across-machine), with no need for communication.
- Load balancing: allocate data to each machine proportionally to its processing power.
- Streaming: add/remove data within-machine, or add/remove machines and update the circular topology accordingly.
- Fault tolerance: revert to older copies of the lost submodels residing in other machines.

In general with multilayer models, e.g. deep nets: ParMAC locks a portion of the input, output and coordinate data $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ inside each machine, and each hidden unit weight vector $\mathbf{w}_h$ is an independent submodel that is communicated in the $\mathbf{W}$ step. Each weight vector may only depend on a subset of the dimensions of $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$, and each dimension of $\mathbf{z}_n$ may only depend on a subset of the weights.



4 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 speedup bottleneck is the number of submodels $M = 2L$ in the $\mathbf{W}$ step.
- The speedups progressively flatten as the number of $\mathbf{W}$-step epochs (and consequently the amount of communication) increases. Using 1–2 epochs gives a good enough result, very close to doing an exact $\mathbf{W}$ step.
- Although these datasets and the number of submodels are relatively small, the speedups we achieve are nearly perfect for $P \leq M$ 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, $e = 2$ 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: $\begin{cases} N\text{: \# training points} \\ M\text{: \# submodels} \end{cases}$