A Discussion on Parallelization Schemes for Vector Quantization Algorithms

Matthieu Durut Benoit Patra Fabrice Rossi

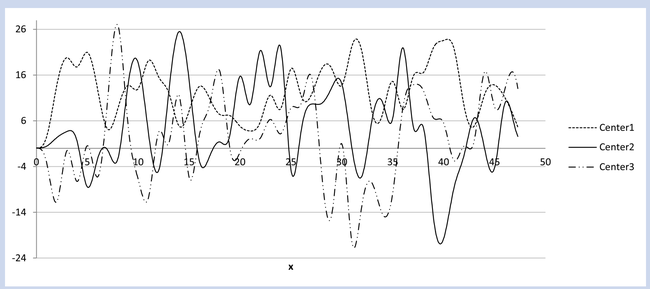
Contribution

We present parallelization schemes for stochastic Vector Quantization (VQ) algorithms in order to obtain time speed-ups using distributed resources. Because of the non-convexity of the VQ loss function, the most intuitive parallelization scheme does not lead to better performances than the sequential algorithm. Other schemes that tackle this issue are proposed and implemented on Microsoft Windows Azure platform obtaining good speed-ups.

Experiment Settings

All the schemes have been tested using simulated distributed architecture. They have been evaluated on a synthetic data set of vector B-Splines using the empirical distortion loss function. For $w \in (\mathbb{R}^d)^\kappa$, it writes:

$$C_{n,M}(w) = \frac{1}{nM} \sum_{i=1}^{M} \sum_{t=1}^{n} \min_{\ell=1,\dots,\kappa} \left\| \mathbf{z}_t^i - w_\ell \right\|^2.$$



Comments

Theoretical results have recently proved that the averaging of local prototypes versions led to the best possible speed-ups (see [1]) under smoothness and convexity assumptions of the loss function. In the VQ case, this function is not convex and the naïve averaging does not lead to any speed-up, as showed in our first experiment.

We provide another distributed scheme, that sums the multiple displacement terms instead of averaging the prototypes versions. This new scheme proves to provide satisfactory speed-ups. It also gives us licence to suggest an asynchronous version of this scheme (as described in equations 3). This latter scheme is suited to implementations on high latency platforms such as cloud computing.

Online and Parallel Online Learning

The VQ technique computes a summary of a dataset $\{\mathbf{z}_t\}_{t=1}^n$ of d dimensional samples with κ prototypes, $w = (w_1, \dots, w_\kappa) \in (\mathbb{R}^d)^\kappa$. The VQ technique belongs to the class of stochastic gradient descent algorithms and compute iterations of the prototypes as follow:

$$w(t+1) = w(t) - \varepsilon_{t+1} H\left(\mathbf{z}^i_{\{t+1 \bmod n\}}, w(t)\right), \quad t \geq 0$$

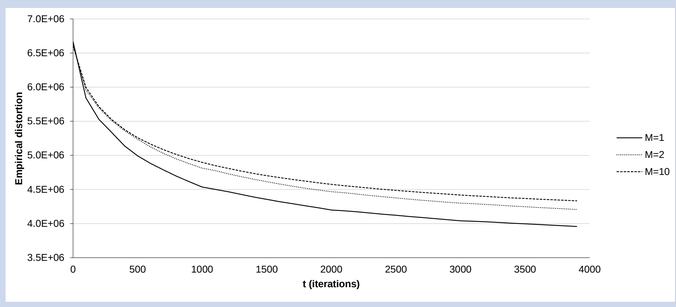
$$with \quad H(\mathbf{z}, w) = \left((w_\ell - \mathbf{z}) \mathbb{1}_{\{l = \operatorname{argmin}_{i=1,\dots,\kappa} \|\mathbf{z} - w_i\|^2\}} \right)_{1 \leq \ell \leq \kappa}, \quad \mathbf{z} \in \mathbb{R}^d \text{ and } w \in (\mathbb{R}^d)^\kappa.$$

We assume having access to M computing entities, each of them executing concurrent VQ procedures by computing prototype versions denoted by $\{w^i(t)\}_{t=0}^\infty$. We investigate techniques that provide a consensus version of the prototypes $\{w^{srd}(t)\}_{t=0}^\infty$ for which the convergence is speeded-up in comparison of the sequential procedure.

Initial scheme

$$\begin{cases} w^i(t) = w^i(t-1) - \varepsilon_t H\left(\mathbf{z}^i_{\{t \bmod n\}}, w^i(t-1)\right), & t \geq 1 \\ \begin{cases} w^{srd}(t) = \frac{1}{M} \sum_{j=1}^{M} w^j(t) \\ w^i(t) = w^{srd}(t) \end{cases} & \text{if } t \bmod \tau = 0. \end{cases} \tag{1}$$

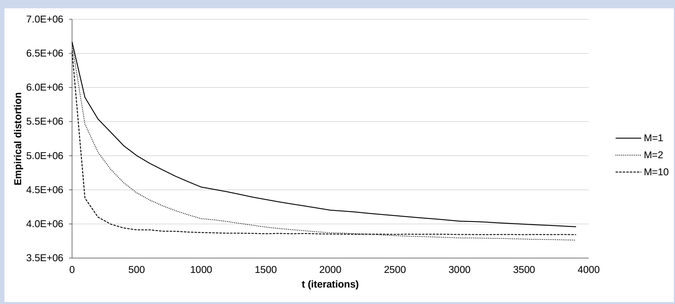
In this scheme, computations are synchronized and prototypes versions are averaged every τ points.



Improved scheme

$$\begin{cases} w^i(t) = w^i(t-1) - \varepsilon_t H\left(\mathbf{z}^i_{\{t \bmod n\}}, w^i(t-1)\right), & t \geq 1 \\ \begin{cases} w^{srd}(t) = w^{srd}(t-\tau) - \sum_{j=1}^{M} \Delta^j_{t-\tau \to t} \\ w^i(t) = w^{srd}(t) \end{cases} & \text{if } t \bmod \tau = 0. \end{cases} \tag{2}$$

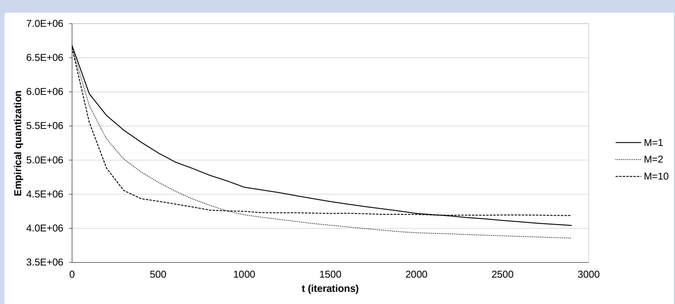
where $\Delta^j_{t_1 \to t_2} = \sum_{t'=t_1+1}^{t_2} \varepsilon_{t'} H\left(\mathbf{z}^j_{\{t' \bmod n\}}, w^j(t'-1)\right)$.



Improved scheme with asynchronism and delays

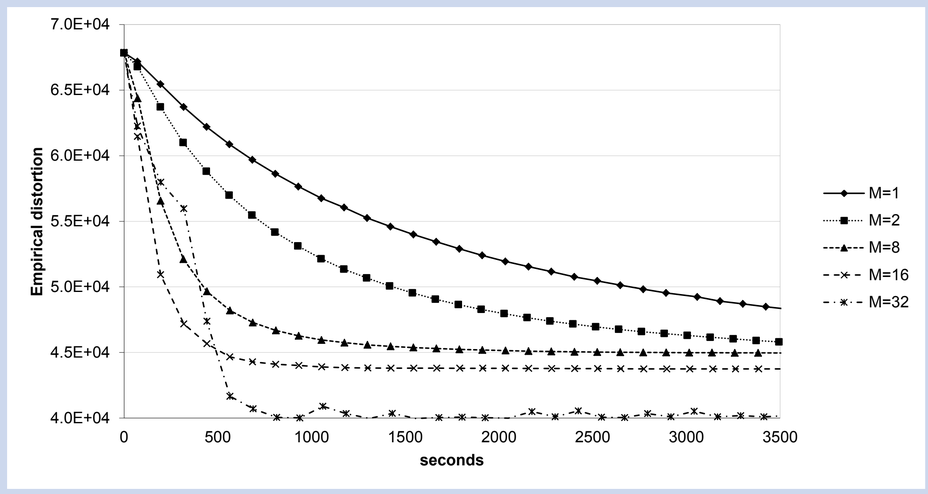
$$\begin{cases} w^i(t) = w^i(t-1) - \varepsilon_t H\left(\mathbf{z}^i_{\{t \bmod n\}}, w^i(t-1)\right), & t \geq 1 \\ w^{srd}(t) = w^{srd}(t-1) - \sum_{j,t=\tau^j(t)} \Delta^j_{\tau^j(\tau^j(t-1)-1) \to \tau^j(t-1)} \\ w^i(t) = w^{srd}(\tau^i(t-1)) - \Delta^i_{\tau^i(t-1) \to t}, & \text{if } t = \tau^i(t) \end{cases} \tag{3}$$

where $\tau^i(t)$ stands for the last time smaller or equal to t for which the computing instance i has shared results with the consensus agent.



Experiments on Windows Azure

The latter scheme has also been implemented on top of the Cloud Computing platform Microsoft Windows Azure. Azure BlobStorage, a No-SQL "key-value pairs" storage, is used as a communication medium between computing units. Snapshots of the consensus shared version are taken regularly and evaluated offline.



Our software project, named CloudDALVQ (the source code is available at http://code.google.com/p/clouddalvq/), makes special care of bottleneck and latency issues related to the consensus building process. This specific design tackle the issue of delays that can significantly reduce convergence speed, as outlined in [2] in the convex case. Our implementation proves to provide satisfactory scale-up up to 32 processing units.

Conclusion

The non-convexity of the VQ loss function forbids to employ consensus techniques such as averaging. The present work provides different distribution schemes that restore satisfactory scale-ups. These latter schemes are adapted to asynchronous implementations that are particularly suited to cloud computing environments. Our cloud implementation on Azure provides scale-ups up to dozens of processing units.

[1] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao Optimal distributed online prediction using mini-batches In Advances in Neural Information Processing Systems 22 2009, p2331-2339

[2] Daniel Hsu, Nikos Karampatziakis, John Langford, Alex J. Smola Parallel Online Learning In Scaling up Machine Learning chapter 14, 2011, p283-306