Distributed deep learning on edge-devices in the Parameter Server Model

Corentin Hardy, Erwan Le Merrer, Bruno Sericola

Distributed deep learning on edge-devices

in the Parameter Server Model

Corentin Hardy, Erwan Le Merrer, Bruno Sericola

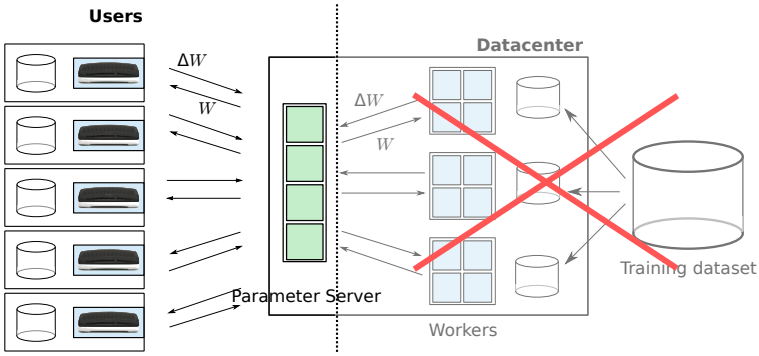
Towards distribution on edge-devices

User devices such as gateways, set-top boxes, smartphones, generate lots of data (voice, pictures, video, network traffic, ...). Most of this data are private and so difficult to exploit. We ask the following question : How to train a deep neural network while keeping users data on their devices ?

The Parameter Server (PS) model allows to distribute SGD iterations :

$$W(t+1) = W(t) + \alpha \Delta W(t)$$

Where W(t) is the state of the machine learning model (such as a deep neural network), and ΔW(t) is the gradient of a loss function computed using a training data batch. In the PS model, W is stored in a server. A pool of workers computes the term ΔW using different training sets. To speed-up the training, workers send their updates and pull the current state of the model from the PS simultaneously asynchronously.
Instead of running the distributed learning in a datacenter, we propose to run workers on edges-devices and to compute updates using their local data.

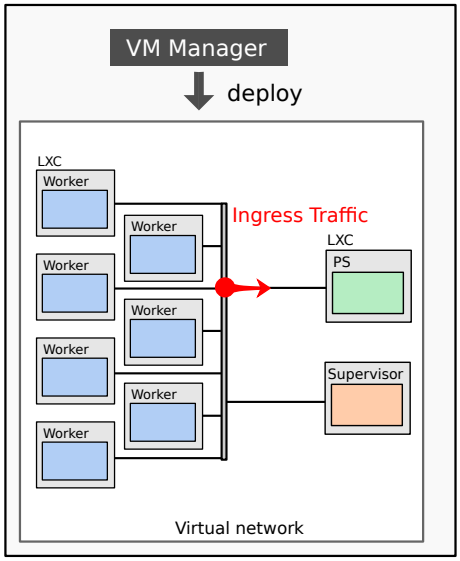


Challenges

In the context of the PS model, deploying workers on edge-devices leads to some issues and challenges :
- The training datasets are composed by user data : how to enforce user privacy ?
- The gradient computation of a deep neural network facing asynchrony and crashes.
- How many workers can train the DNN in parallel ?
- Instead of datacenter communication, updates are sent through the Internet, with bandwidth or latency constraints.

Ingress traffic

The ingress traffic is a critical bottleneck : all workers send their updates to the PS located on a central node. Classically, updates ΔW have the same size than the model. In case of deep learning this mean the central node has to receive between few MB to GB per worker each time one computes an update. This bottleneck limits parallelism and consequently the performance of the training.



Our experimental plateform

Our proposal : AdaComp

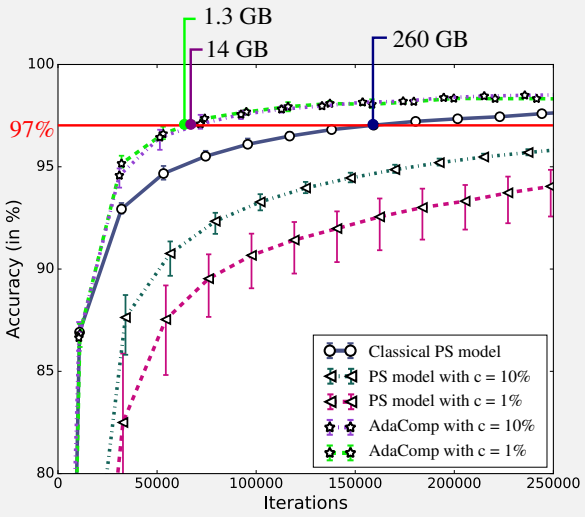
To deal with the ingress traffic bottleneck, we propose Adacomp, a method which combines gradient selection to send sparse updates and an adapted learning rate using element-staleness. The selection is performed by keeping gradient of parameters with the largest magnitude. For each variable of the model (i.e., matrix weights in the case of DNN), workers send gradients of only a fraction c of its total number of elements (selected at each iterations).
The staleness is computed locally for each parameter of the model. At iteration i, the staleness for the parameters k is computed such as :

$$\sigma_k(i) = \sum_{u=i-d}^{i-1} \mathbb{1}_{\{\Delta W_k(u) \neq 0\}}$$

where $\Delta W_k(u)$ is the gradient of parameter k at iteration u, and d is the delay of current state of worker. The staleness is used to adapt the learning rate of each parameter :

$$\alpha_k(i) = \begin{cases} \alpha/\sigma_k(i) & \text{if } \sigma_k(i) \neq 0 \\ \alpha & \text{otherwise.} \end{cases}$$

where α is the learning rate. Thus, parameters which are rarely updated since the last iteration of the worker obtain an higher learning rate than parameters which are often updated. AdaComp allows to reduce ingress traffic at PS while keeping a good accuracy.



Test accuracy of an distributed CNN on MNIST with 200 workers