Neurocomputing xxx (xxxx) xxx

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Distributed optimization for deep learning with gossip exchange

Michael Blot[a,1,*], David Picard[b,a], Nicolas Thome[a,c], Matthieu Cord[a]

[a] UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, Sorbonne Universités, 4 place Jussieu, Paris 75005, France
[b] ETIS UMR 8051, Université Paris Seine, Université Cergy-Pontoise, ENSEA, CNRS, France
[c] CEDRIC, Conservatoire National des Arts et Métiers, 292 Rue St Martin, Paris 75003, France

ARTICLE INFO

Article history:
Received 1 February 2018
Revised 3 October 2018
Accepted 2 November 2018
Available online xxx

Communicated by Zechao Li

Keywords:
Optimization
Distributed gradient descent
Gossip
Deep
Learning
Neural networks

ABSTRACT

We address the issue of speeding up the training of convolutional neural networks by studying a distributed method adapted to stochastic gradient descent. Our parallel optimization setup uses several threads, each applying individual gradient descents on a local variable. We propose a new way of sharing information between different threads based on gossip algorithms that show good consensus convergence properties. Our method called GoSGD has the advantage to be fully asynchronous and decentralized.

1. Introduction

With deep convolutional neural networks (CNN) introduced by Fukushima [1] and LeCun et al. [2], computer vision tasks and more specifically image classification have made huge improvements in the years following [3]. CNN performances benefit a lot from big collections of annotated images like Russakovsky et al. [4] or Lin et al. [5]. They are trained by optimizing a loss function with gradient descents computed on random mini-batches according to Bottou [6]. The method called stochastic gradient descent (SGD) has proved to be very efficient to train neural networks in general.

However current CNN structures are extremely deep like the 100 layers ResNet of He et al. [7] and contains a lot of parameters (around 60 M for Alexnet [3] and 130 M for vgg [8]). Those structures involve heavy gradient computation times making the training on big data-sets very slow. Computation on GPU accelerates the training but requires huge local memory caches.

Nevertheless the mini-batch optimization seems suitable for distributing the training over several threads. Many methods have been studied like Refs. [9,10], which propose to distribute the batches over different threads called workers that periodically exchange information via a central thread to synchronize their models. In [9], when a worker exchanges information with the central node it updates its parameters variable with a weighted averaging of its own parameters variable and the central node ones. The master thread does the symmetrical update. In [10], the workers accumulate in an additional buffer the gradients computed at the last steps, and applies this aggregated gradient to the master thread parameters.

The major issue of these methods is that they induce waits in all computing nodes due to the synchronization problems with the critical resource that is the central node. A popular way of tackling this issue in distributed optimization involves exchanging information in a peer to peer fashion. In this category, Gossip protocols [11] have been successfully applied to many machine learning problems such as kernel methods [12], PCA [13] and k-means clustering [14,15]. We propose to combine Gossip protocols with SGD to efficiently train deep CNN without the need of a central node.

The contributions of this paper are the following. First, we propose a new framework for distributed SGD in which we can formally analyze and compare the properties of existing distributed deep learning training algorithms. Using this framework, we show that distributing the mini-batches is equivalent to use bigger batches in non-distributed SGD, which has implications on the convergence speed of the algorithms. Finally, by carefully analyzing the key aspects of our framework, we propose two distributed

* Corresponding author.
E-mail addresses: michael.blot@lip6.fr (M. Blot), picard@ensea.fr (D. Picard), nicolas.thome@lip6.fr (N. Thome), matthieu.cord@lip6.fr (M. Cord).
[1] Thank *DGA* for financing my researches.

https://doi.org/10.1016/j.neucom.2018.11.002
0925-2312/

SGD algorithms for training deep CNN. The first one, called PerSyn, uses a central node in a very efficient way. The second one, called GoSGD, is based on Gossip exchanges and can thus be scaled to an arbitrarily large number of nodes, providing a significant speedup to the training procedure.

The remaining of this paper is as follows. In the next section, we recall the principles of SGD applied to deep learning as well as the relevant literature about distributing its computation. Next, in Section 3, we detail our formalization of distributed SGD which allows use to compare existing algorithms and write our first proposition PerSyn. In Section 4, we present a decentralized alternative based on Gossip algorithms, that we call GoSGD, which fits nicely in our framework. Finally, we present experiments showing the efficiency of distributing SGD in Section 5 before we conclude.

2. SGD and related work

In image classification, training deep CNN follows the empirical risk minimization (ERM) principle [16]: The objective is to minimize

$$L(x) = \mathbb{E}_{Y\sim\mathcal{I}}[\ell(x, Y)] \tag{1}$$

where Y is a pair variable consisting of an image and the associated label following the natural image distribution $\mathcal{I}$, x are the CNN parameters and ℓ the loss function quantifying the prediction error made on Y. Despite the objective function usually being non convex, gradient descent has been shown to be a successful optimization method for the problem. In gradient descent algorithm [17], the update operation

$$x \leftarrow x - \eta\nabla L(x) \tag{2}$$

is sequentially applied to the optimized variables until a convincing local minimum for L is found or some stopping criteria are reached. η is the gradient step size, called learning rate. In image classification, it is not possible to compute the expected gradient $\nabla L(x)$ because the distribution $\mathcal{I}$ is unknown (and not properly modeled). However it is possible to approximate it using a Monte Carlo method: $\nabla L(x) = \nabla_x\Big(\mathbb{E}_Y[\ell(x, Y)]\Big) = \mathbb{E}_Y[\nabla_x\ell(x, Y)]$ (the properties allowing to swap the derivative and the integral are supposed to be respected). This last quantity is approximated by a Monte Carlo estimator:

$$\widehat{\nabla}L(x) = \frac{1}{N}\sum_{i=1}^{N}\nabla_x\ell(x, Y_i)$$

The set $(Y_i)_{i=1..N}$ of independent samples drawn from $\mathcal{I}$ is called a batch and N is the batch size. $\widehat{\nabla}L(x)$ is then the descent direction used at each update. In practice, the images in a batch are drawn randomly from the training set and are different for each gradient step.

Concerning the precision of the gradient indicator, we point out that it is unbiased. It can also be shown that the approximation with respect to true gradient depends on the batch size. In fact, $\mathbb{E}||\nabla L(x) - \widehat{\nabla}L(x)||_2^2 \propto \frac{1}{N}tr(\mathbb{C}ov_Y[\nabla\ell(x, Y)])$ where $\mathbb{C}ov_Y[\nabla\ell(x, Y)]$ is the covariance matrix of the gradient estimator and tr denotes the trace of the matrix (see Appendix A). Therefore the bigger the batch size N, the more accurate the gradient estimator. Unfortunately GPU memory is limited and bounds the number of images that can be processed in a batch. For instance, a GPU card with 12 Go of memory cannot process batches bigger than 48 images of 224×224 RGB pixels using Resnet [7] with 30 layers. Indeed, at each update step, the GPU needs to host the whole model intermediate computations for all images in the bat ch. As such, using a single GPU limits the precision of the stochastic gradient descent and thus hinders its converge nce.

2.1. Related work

There are many ways to accelerate the convergence time of SGD, one of which is to optimize the learning rate η for each gradient descent iteration as aimed at by second order gradient descent methods [17]. For deep learning, the parameters space is of high dimension and second order methods are computationally too heavy to be applied efficiently. Examples of first order gradient descent that adapt the learning rate with success can be found in [18] and [19].

A second way to accelerate the gradient descent is to reduce the computation time of $\widehat{\nabla}L(x)$ which is the purpose of this paper. Based on the availability of several GPU cards, different methods that overcome the GPU batch size limitations have been proposed. There are two main ways of distributing the computation of the gradient update, which are either splitting the model and distributing parts of it among the different GPUs, or splitting the batch of images in subsets and distributing them among the GPUs.

A famous technique used by Krizhevsky et al. [3] for distributing the model consists in paralleling the forward/backward pass of a batch over two GPU cards. The convolutional part of the network is split into two independent parts that are only aggregated with a fully connected layer at the top of the network. This conception enables to load the two parallel parts on two different GPUs liberating space for more images in batches. However the CNN models claiming current state of the art performances in image classification like He et al. [7] and Huang et al. [20] benefit a lot from the inter-correlation between features of each layers and do not allow independent partitioning within a layer. Thus if the network is loaded in several GPU cards they would have to communicate during the forward pass of a batch. The communications being time-consuming it is preferable to restrain the whole network on a single GPU card.

The second approach, which consists in distributing subsets of the batch to the different GPUs, is becoming increasingly popular. We use the following notations to describe such methods: As in [9] the different threads that manage a neural network model are called workers. With M GPU cards, there are M workers and their corresponding set of parameters is noted x_m for each worker m. The set of parameters used for inference is noted $\tilde{x}$ and is specifically the model we want to optimize. The easiest way of distributing the batches is presented in [21] and consists in computing the average of all the workers models as in Algorithm 1.

Algorithm 1 Fully synchronous SGD: pseudo-code.

Input: M: number of threads, η : learning rate
Initialize: x is initialized randomly, $x_m = x$
repeat
  $\forall m,\ v_m = \widehat{\nabla}L_m(x_m)$
  $v = \frac{1}{M}\sum_{m=1}^{M} v_m$
  $\forall m, x_m = x_m - \eta v$
until Maximum iteration reached
return $\frac{1}{M}\sum_{m=1}^{M} x_m$

We stress the fact that after each iteration all workers host the same value of the variable $\tilde{x}$ to optimize (i.e., $\forall m, x_m = \tilde{x}$). This method is equivalent to using M times bigger batches for the gradients computation.[2] It is then possible to alleviate the GPU memory constraint limiting the number of examples in a batch in order to

[2] In [21] they use the sum instead of the averaging of the gradients but both are equivalent by adjusting the learning rate.

Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002

get more accurate gradients. However in comparison with the classical SGD there are three additional phases of communication and aggregation that have incompressible time:

1. The transmission to the master of the local gradient estimates computed by the threads at line 4.
2. The aggregation of the gradients at line 5.
3. The transmission of the aggregated gradient $\hat{\nabla}L(x)$ to all threads before local updates at line 6.

In large scale image classification, the communications involved at lines 4 and 6 imply a CNN with millions of parameters and are thus very costly. Moreover, to perform line 5, the master has to wait for all workers to have completed line 4. The additional time involved by communications and synchronizations can have a significant impact on the global training time. This is even more true when the GPUs are not on the same server, where the communication delays are consequent. Thus, this easy synchronous distributed method can be very inefficient.

The focus of this paper is to reduce these communication and aggregation costs. As with existing strategies such as Refs. [9,10], the main idea is to control the amount of information exchanged between the nodes. This amount is a key factor in making all workers contribute to the optimization of the test model $\tilde{x}$. Indeed, if no information is ever exchanged, the distributed system is equivalent to training M independent models. As a consequence of the symmetry property of CNN explained in [22], these independent models are likely to be very different and almost impossible to combine. As such, the distributed training would not improve over using a single GPU with a reduced batch size. The main challenge of this work is thus to allow as little information exchange as possible to ensure reduced communication costs, while still optimizing the combined model $\tilde{x}$.

In the next section, we introduce a matrix framework that allows to explicitly control the amount of exchanged information, and thus to explore different optimization strategies.

3. Framework

In this section, we propose a general framework for distributed SGD methods. We start from the centralized SGD update procedure and we show the equivalent synchronous distributed procedure. Then, we show that relaxing some equality constraints among workers is equivalent to consider different communication strategies. We are thus able to write strategies found in the literature using a simple matrix expression.

First, we unroll the recursion of the classic SGD update of Eq. (2) to obtain the value of $\tilde{x}$ after $T+1$ gradient descent steps:

$$\tilde{x}^{(T+1)} = \tilde{x}^{(0)} - \eta \sum_{t=0}^{T} \hat{\nabla} L\left(\tilde{x}^{(t)}\right) \tag{3}$$

With $\tilde{x}^{(t)}$ being variable $\tilde{x}$ a time t. The equivalent batch distribution method is the following:

$$\tilde{x}^{(T+1)} = \tilde{x}^{(0)} - \frac{\eta}{M} \sum_{t=0}^{T} \sum_{m=1}^{M} \hat{\nabla} L_m\left(\tilde{x}^{(t)}\right)$$

In order to make the communication between the workers and the central model more visible, we can use the equivalent update rule that introduces local variables x_m and consensus constraints. By doing so, we obtain the standard distributed SGD method:

$$\tilde{x}^{(T+1)} = \tilde{x}^{(0)} - \frac{\eta}{M} \sum_{t=0}^{T} \sum_{m=1}^{M} \hat{\nabla} L_m\left(x_m^{(t)}\right) \tag{4}$$

$$\text{s.t. } \forall t, \forall m, x_m^{(t)} = \tilde{x}^{(t)} \tag{5}$$

We can replace the consensus equality constraints (5) by equivalent local update rules obtained from (4):

$$\forall r, x_r^{(T+1)} = \tilde{x}^{(0)} - \frac{\eta}{M} \sum_{t=0}^{T} \sum_{s=1}^{M} \hat{\nabla} L_s\left(x_s^{(t)}\right)$$

This formulation is a first step towards decentralized and communication efficient algorithms since it made the communications between workers visible. Indeed, at each step, each worker gathers the gradients from all other workers and applies them to its local variable. The main drawback of this formulation is that the number of messages exchanged by workers at each step is now $M(M-1)$ instead of $2M$ in the classical version.

To allow for more efficient communication, we relax the consensus constraints by allowing local variables to differ slightly. This can be done efficient by introducing non-uniform weights $\alpha_{r,s}$ in the combination of the local gradients:

$$\tilde{x}^{(T+1)} = \tilde{x}^{(0)} - \eta \sum_{t=0}^{T} \sum_{m=1}^{M} \alpha_{0,m} \hat{\nabla} L_m\left(x_m^{(t)}\right)$$

$$\forall r, x_r^{(T+1)} = \tilde{x}^{(0)} - \eta \sum_{t=0}^{T} \sum_{s=1}^{M} \alpha_{r,s} \hat{\nabla} L_s\left(x_s^{(t)}\right)$$

$$\text{s.t. } \forall r, \sum_{s=1}^{M} \alpha_{r,s} = 1$$

We introduce a matrix notation over $\alpha_{r,s}$ to simplify the expression of these update rules. We note $\mathbf{x}^{(t)} = [\tilde{x}^{(t)}, x_1^{(t)}, \ldots, x_M^{(t)}]$ the vector concatenating the central variable $\tilde{x}^{(t)}$ and all local variables $x_m^{(t)}$ at time step t. Similarly, we note $\mathbf{v} = [0, \hat{\nabla} L_1(x_1^{(t)}), \ldots, \hat{\nabla} L_M(x_M^{(t)})]$ the vector concatenating all the local gradients $\hat{\nabla} L_m(x_m^{(t)})$ at time step t, with a leading 0 to have the same size as $\mathbf{x}$. The weights $\alpha_{r,s}$ of the gradient combinations are enclosed in a matrix K, which leads to the following matrix update rule :

$$\mathbf{x}^{(T+1)} = \left(\prod_{t=0}^{T} K\right) \mathbf{x}^{(0)} - \eta \sum_{t=0}^{T} \left(\prod_{\tau=t}^{T} K\right) \mathbf{v}^{(t)}$$

K is the communication matrix and is responsible for the mixing of the local updates. Remark that its rows have to sum to 1 to avoid exploding gradients. The sparser K is, the less workers exchange information and thus the more efficient the algorithm is. However, this comes at the price of less consensus among the local models. On the extreme, if K is diagonal (no information exchanged between workers), these update rules are equivalent to M different model being independently trained. Hence, the choice of K is critical to have a good trade-off between low communication costs and good information sharing between workers.

To further allow for the optimization of K, we introduce time dependent communication matrices $K^{(t)}$. This allows for the design of very sparse communication matrices most of the time, balanced with the use of a dense $K^{(t)}$ to enforce better consensus whenever it is required. We note $P_t^T = \prod_{\tau=t}^{T} K^{(\tau)}$, and obtain:

$$\mathbf{x}^{(T+1)} = P_0^T \mathbf{x}^{(0)} - \eta \sum_{t=0}^{T} P_t^T \mathbf{v}^{(t)}$$

This corresponds to the following recursion:

$$\mathbf{x}^{(T+1)} = K^{(T)} \mathbf{x}^{(T)} - \eta K^{(T)} \mathbf{v}^{(T)}$$

Remark that since the update is linear, mixing the local gradients and mixing the local variables are equivalent. As such, we can define an intermediate variable $\mathbf{x}^{(t+\frac{1}{2})} = \mathbf{x}^{(t)} - \eta \mathbf{v}^{(t)}$ that

Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002

considers only the local updates and obtain the following update rules:

$$\mathbf{x}^{(T+\frac{1}{2})} = \mathbf{x}^{(T)} - \eta \mathbf{v}^{(T)} \tag{6}$$

$$\mathbf{x}^{(T+1)} = K^{(T)} \mathbf{x}^{(T+\frac{1}{2})} \tag{7}$$

Since these rules make a clear distinction between local computation (step $T+\frac{1}{2}$) and communication (step $T+1$), they are the ones we will use in the remaining of this article.

In the following, we show how several existing distributed SGD can be specified by their sequence of communication matrices $K^{(t)}$. We first start with a naive yet communication efficient method we call *PerSyn* (Periodically Synchronous) that considers exchanging information only once every few steps. Then, we show how EASGD [9] and DOWNPOUR [10] compare to PerSyn.

3.1. PerSyn

As a first illustration of our framework we introduce a new communication and aggregation strategy for distributed SGD that is very simple to implement and yet surprisingly effective. The main idea is to relax the synchronization of all local variables with the master variable to occur only once every τ steps. This is done by defining the following time dependent communication matrices:

$$K^{(t)} = \begin{cases} \begin{pmatrix} 0 & \frac{1}{M}\mathbb{1} \\ \vdots & I \end{pmatrix}, & \text{if } t \bmod \tau = 0 \\ \begin{pmatrix} 0 & \cdots \\ \mathbb{1} & 0 \end{pmatrix}, & \text{if } t \bmod \tau = 1 \\ \begin{pmatrix} 0 & \cdots \\ \vdots & I \end{pmatrix}, & \text{else} \end{cases}$$

I being the identity matrix of size $M \times M$, and $\mathbb{1}$ the vector of size M with 1 on every component. The corresponding algorithm is described in Algorithm 2. Remark it is very similar to the standard distributed SGD (1) and thus can be very easily implemented.

Algorithm 2 PerSyn SGD: Pseudo-code.

Input: M: number of threads, η : learning rate
Initialize: x is initialized randomly, $x_m = x$, $t = 0$
repeat
  for all m, $x_m^{(t+\frac{1}{2})} \leftarrow x_m^{(t)} - \eta \widehat{\nabla} L_N^m(x_m^{(t)})$
  $t = t + 1$
  if $t \bmod \tau = 0$ **then**
    $\tilde{x}^{(t+1)} = \frac{1}{M}\sum_{m=1}^{M} x_m^{(t+\frac{1}{2})}$
    $\forall m$, $x_m^{(t+1)} \leftarrow \frac{1}{M}\sum_{m=1}^{M} x_m^{(t+\frac{1}{2})}$
  else
    $\forall m$, $x_m^{(t+1)} \leftarrow x_m^{(t+\frac{1}{2})}$
  end if
until Maximum iteration reached
return $\frac{1}{M}\sum_{m=1}^{M} x_m$

As we can see, PerSyn involves no communication at all $\frac{\tau-1}{\tau}$ percent of the time. However, during that time, models are driven by their local gradients only and can thus diverge from one another leading to incoherent distributed optimization. The trade-off with PerSyn is then to choose τ such as to minimize the communication costs while forbidding models to diverge from one another during the time where no communication is made.

One main limitation of PerSyn is that the communication matrix is very dense when $t \bmod \tau = 0$. This is taken care of by EASGD [9] by considering a moving average of the local models instead of a strict average.

3.2. EASGD

In EASGD, the local models are periodically averaged like in PerSyn. However, the averaging is perform in an elastic way that allows for reduced communication compared to PerSyn. The corresponding matrix is then:

$$K^{(t)} = \begin{cases} \begin{pmatrix} 1-M\alpha & \alpha\mathbb{1} \\ \alpha\mathbb{1} & (1-\alpha)I \end{pmatrix}, & \text{if } t \bmod \tau = 0 \\ \begin{pmatrix} 0 & \cdots \\ \vdots & I \end{pmatrix}, & \text{else} \end{cases}$$

With α being a mixing weight.

As we can see, most of the time, no communication is performed and the communication matrix is the identity. Every τ iterations, an averaging synchronization is performed to keep the local models from diverging from one another. However, contrarily to PerSyn which replaces all models (local and global) by the new average, this is done by computing a weighted average between the previous model and the current average. By doing so, EASGD is able to exchange only $2M$ messages every τ iteration, without having to wait for the central node to compute the new average. However, a global synchronization is still required as the master has to perform the weighted average of local models that are consistent, that is, local models that have been updated the same number of time.

3.3. Downpour SGD

In Downpour SGD [10], an asynchronous update scheme is discussed. By asynchronous, the authors mean that each worker is endowed with its own clock and process gradient at a rate that is completely independent from the other workers. This can be easily described in our matrix framework by considering a universal clock that ticks each time one of the workers clock ticks. This corresponds to considering the finest time resolution possible, when only one worker is awake at any given time step. In particular, this redefines $\mathbf{v}^{(t)} = [0, \ldots, 0, \widehat{\nabla} L_m(x_m^{(t)}), 0, \ldots]$ to have zeros everywhere but on the component corresponding to the awaken worker m.

Whenever they awake, each worker has the option to communication with the master while performing its gradient update. These communications can either be fetching the global model from the master or sending the current update to the master. This is expressed by the following communication matrices:

$$K^{(\text{send})} = \begin{pmatrix} 1 & \mathbf{e}_m \\ 0 & I \end{pmatrix}, \quad K^{(\text{receive})} = \begin{pmatrix} 1 & 0 \\ \mathbf{e}_m & I - \mathbf{e}_m\mathbf{e}_m^\top \end{pmatrix}$$

with $\mathbf{e}_m$ being the vector of zeros with a 1 on component m.

When nodes are neither sending nor receiving, the communication matrix is simply the identity and thus the only ongoing operation is a computation which involves a single worker that performs a local gradient update. The main drawback of Downpour is that it involves a master that stores the most up-to-date model. This single entry point can be a source of weaknesses in the the training process since it acts as the communication bottleneck (all workers have to communicate with the master) and is also a critical point of failure. We intend to overcome this drawback by introducing new communication matrices that lead to fully decentralized training algorithms.

Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002

4. Gossip Stochastic Gradient Descent (GoSGD)

To remove the burden of using a master, several key modifications to the communication process have to be made. First, the absence of a master means that each worker is expected to converge to the same value $\tilde{x}$, which corresponds to ensuring a strict consensus among workers. Furthermore, this absence of a master is reflected by the communication matrix having its first row and first column to be 0. Therefore, all communication are performed in a peer to peer way, which is reflected in the communication matrix where the columns are the senders and the rows are the receivers.

Second, the absence of a master also implies the absence of a global clock. This means that workers are performing their updates at any time, independently of the others. To reflect this, we consider the same clock model as with Downpour, where at each time step t, only a single worker s is awaken. Consequently, this also redefines $\mathbf{v}^{(t)} = [0, \dots, 0, \hat{\nabla} L_s(x_s^{(t)}), 0, \dots]$ to have zeros everywhere but on the component corresponding to the awaken worker s. Furthermore, since the active worker is random, the communications have to be randomized too, which leads to random communication matrices $K^{(t)}$. These random peer to peer communication systems are known as Randomized Gossip Protocols [11], and are known to converge to the consensus exponentially fast in the absence of perturbations (which in our case corresponds to $\forall t, \mathbf{v}^{(t)} = 0$). To control the communication cost, the frequency at which a worker emits messages is set by using a random variable deciding whether the awaken node shares its variables with neighbors.

Finally, we want a communication protocol where no worker is waiting for another, so as to maximize the time they spend at computing their local gradients. Waiting occurs when a worker is expecting a message from another worker and has to idle until this message arrives. This is the case in symmetrical communication where workers expect replies to their messages. It is well known in the Randomized Gossip literature that such local blocking waits can cause global synchronization issues where most of the workers spend their time waiting for the needed resources to be available. To overcome this problem, asymmetric gossip protocols have been developed such as in [23]. Asymmetric means that no worker is both sending and receiving messages at the same time, which translates in constraints on the non-zero coefficients of the communication matrix $K^{(t)}$. To assure convergence to the consensus, a weight $w_m^{(t)}$ has to be associated with every variable $x_m^{(t)}$ and is shared by workers using the same communications as the variables $x_m^{(t)}$. These communication protocols are known as Sum-Weight Gossip protocols because of introduction of the weights $w_m^{(t)}$.

Using these assumptions, we define the Gossip Stochastic Gradient Descent (GoSGD) as follows: Let s be the worker awaken at time t, which is our potential sender. Let $S \sim B(p)$ be a random variable following a Bernoulli distribution of probability p. Let r be a random variable sampled from the uniform distribution in $\{1, \dots, M\} \setminus s$. r represents our potential receiver. The communication matrix $K^{(t)}$ is then:

$$K^{(t)} = \begin{cases} \begin{pmatrix} 0 & \cdots \\ \vdots & I + \frac{w_s^{(t)}}{w_s^{(t)}+w_r^{(t)}} \mathbf{e}_r \mathbf{e}_s^\top + \left(\frac{w_s^{(t)}}{w_s^{(t)}+w_r^{(t)}} - 1 \right) \mathbf{e}_s \mathbf{e}_s^\top \end{pmatrix}, & \text{if } S \\ \begin{pmatrix} 0 & \cdots \\ \vdots & I \end{pmatrix}, & \text{else} \end{cases} \tag{8}$$

If S is successful, the weights of the sender s and the receiver r are updated as follows:

$$w_s^{(t+1)} = \frac{w_s^{(t)}}{2}, \quad w_r^{(t+1)} = w_r^{(t)} + \frac{w_s^{(t)}}{2} \tag{9}$$

Notice that this update involves a single message form s to r just as for the update of $x_r^{(t)}$ using $K^{(t)}$. In practice, both $x_s^{(t)}$ and $w_s^{(t)}$ are encapsulated in a single message and sent together.

Remark also that contrarily to what is found in the sum-weight gossip literature, we do not perform explicit ratio of the variables and their associated weights. Our implicit ratio is performed by including the weights in the communication matrix $K^{(t)}$. This leads to more complicated matrices than what is common in the literature and hinders the theoretical analysis of the communication process (since $K^{(t)}$ are no longer i.i.d.). However, it also leads to a much easier implementation when considering large deep neural networks since the gradient is computed on the variable only and not on the ratio of variables and associated weights. Furthermore, since the updates are equivalent, our proposed communication matrix keeps all the exponential convergence to consensus properties of standard sum-weights gossip protocols.

4.1. GoSGD algorithm

The procedure run by each worker to perform GoSGD is described in Algorithm 3. Each worker is endowed with a queue q_m which can be concurrently accessed by all workers. Each worker repeats the same loop which consists in a sequence of 3 operations, namely processing incoming messages, performing a local gradient descent update and possibly sending a message to a neighbor worker.

Algorithm 3 GoSGD: worker Pseudo-code.

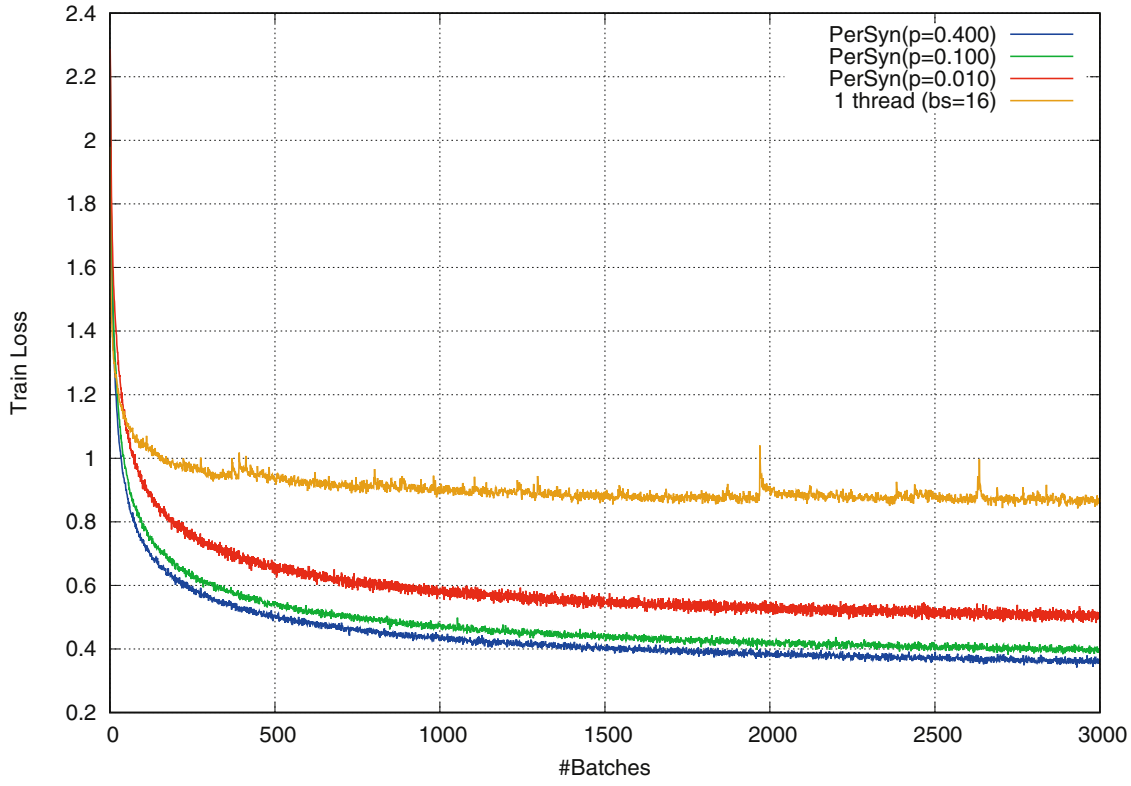
Input: p: probability of exchange, M: number of workers, η : learning rate, $\forall m, q_m$: message queue associated with worker m, s current worker id.
Initialize: x is initialized randomly, $x_s = x$, $w_s = \frac{1}{M}$
repeat
  processMessages(q_s)
  $x_s \leftarrow x_s - \eta^{(t)} \hat{\nabla} L_s^{(t)}$
  if $S \sim B(p)$ **then**
    $r = Random(M)$
    pushMessage(q_r, x_s, w_s)
  end if
until Maximum iteration reached
return x_s

Algorithm 4 Gossip update functions.

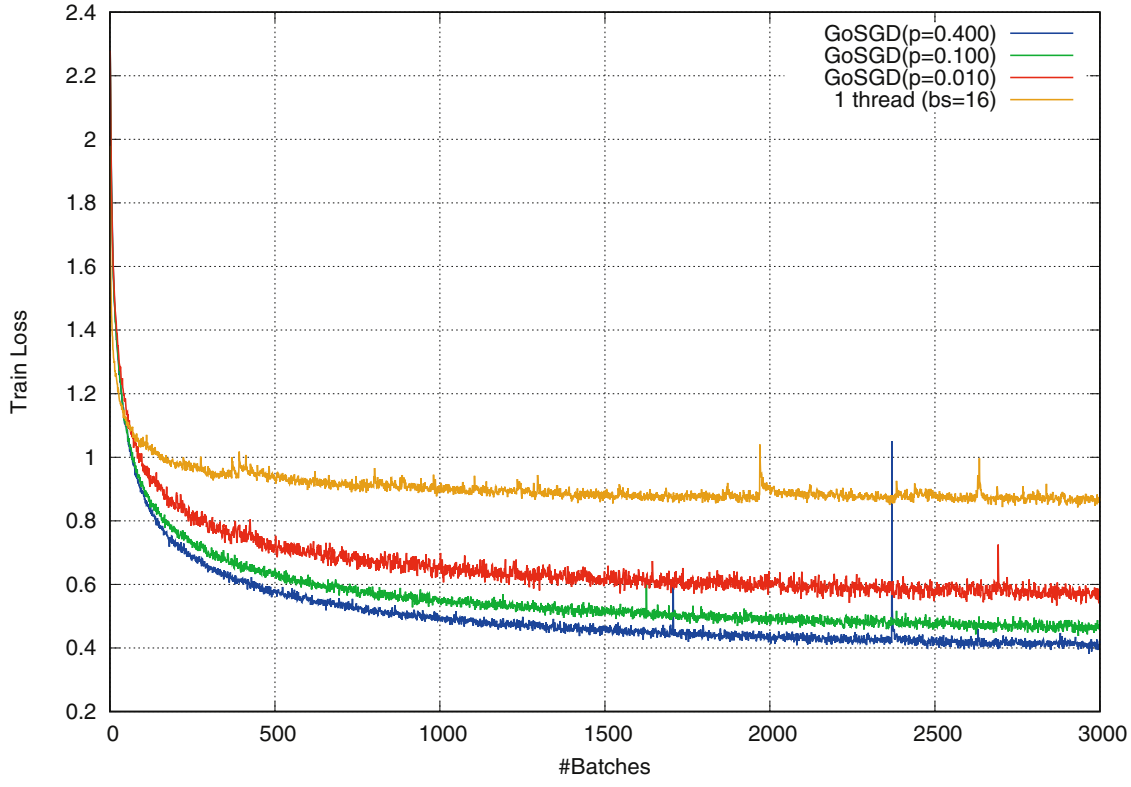
function pushMessage(queue q_r, x_s, w_s)
  $x_s \leftarrow x_s$
  $w_s \leftarrow \frac{w_s}{2}$
  q_r.push((x_s, α_s))
end function
function processMessages(queue q_r)
  repeat
    $(x_s, w_s) \leftarrow q_r$.pop()
    $x_r \leftarrow \frac{w_r}{w_s+w_r} x_r + \frac{w_s}{w_s+w_r} x_s$
    $w_r \leftarrow w_r + w_s$
  until q_r.empty()
end function

Remark that all messages are processed in a delayed fashion in the sense that several messages can be received and the corresponding variables may have been updated by their respective workers before the receiving worker applies the reception procedure. From this point of view, it seems the workers are taking into account outdated information, which is even more the case with a low communication frequency controlled by a low probability

Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002



(a) Training loss evolution for PerSyn



(b) Training Loss evolution for GoSGD

Fig. 1. Training loss evolution for PerSyn and GoSGD for different frequency/probability of exchange p.

Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002

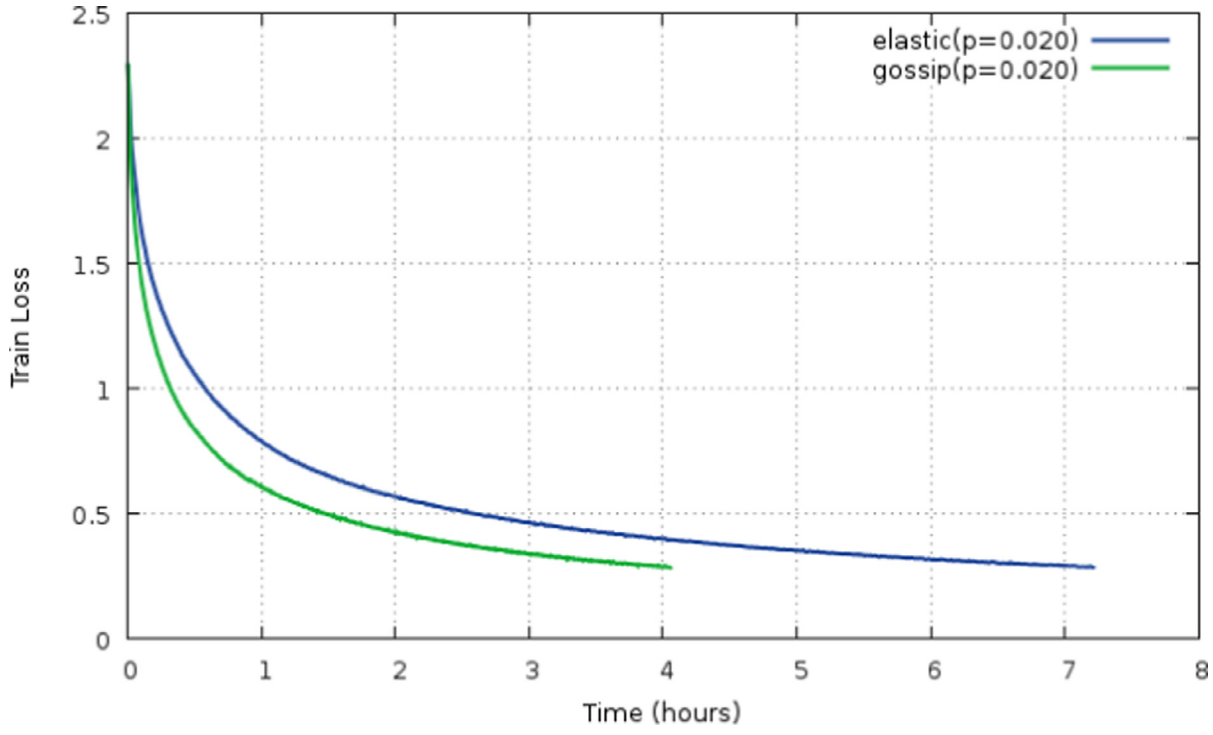


Fig. 2. Convergence speed comparison between GoSGD and EASGD for similar frequency/probability of exchange $p = 0.02$.

p. However, as we show in the experiment, even very low probabilities of communication ($p = 0.01$) yield very good convergence properties and satisfying consensus.

4.2. Distributed optimization interpretation

We now show that our proposed GoSGD solves a consensus augmented version of the distributed ERM principle. As shown in Section 3, the easiest way of tackling the distributed ERM principle is to consider local variables constrained to be equal:

$$\min_{x_1,\dots,x_M} \sum_{m=1}^{M} L(x_m)$$
$$\text{s.t. } \forall m, x_m = \hat{x}$$
$$\hat{x} = \frac{1}{M}\sum_{m=1}^{M} x_m$$

As used in [9] and [24], the consensus constraints can be relaxed which leads to the following augmented problem:

$$\min_{x_1,\dots,x_M} \sum_{m=1}^{M} L(x_m) + \frac{\rho}{2}\|x_m - \hat{x}\|_2^2 \tag{10}$$

$$\text{s.t. } \hat{x} = \frac{1}{M}\sum_{m=1}^{M} x_m \tag{11}$$

We can rewrite this loss in order to make the equivalent gossip communications visible:

$$\min_{x_1,\dots,x_M} \sum_{s=1}^{M} \left(L(x_s) + \frac{\rho}{4M}\sum_{r=1}^{M} \|x_s - x_r\|_2^2 \right) \tag{12}$$

The gradient of this objective function contains 2 terms, the first being related to the empirical risk and the second being the pairwise distance between local models. We prove in Appendix B that the update rules of GoSGD are in expectation equivalent to this two gradient terms and thus that GoSGD performs a stochastic alternate gradient descent on this augmented problem.

5. Experiments

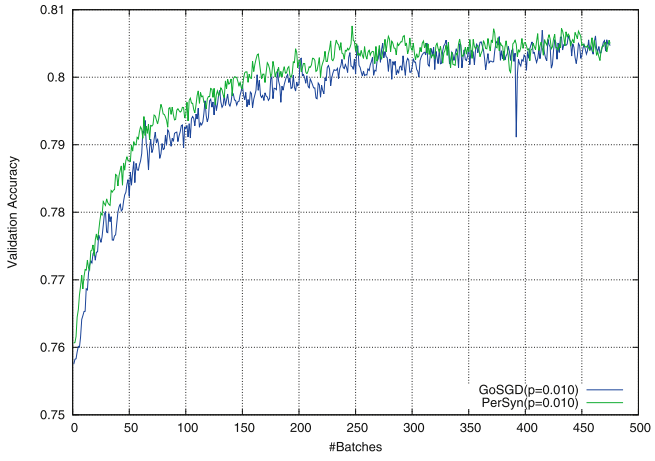
We conducted experiment on the CIFAR-10 dataset (see [25] for detailed presentation) using a CNN defined in [9] and described in [26]. In the first part, we test the convergence rate of different distributed algorithms, namely PerSyn, GoSGD and EASGD. The second part of our experiments considers the consensus convergence ability of the different communication strategies. For distributed experiments we always use $M = 8$ workers loaded on 8 different Tesla K-20 GPU cards with 5 Go of RAM each. The deep learning framework that we used is Torch 7 [27]. For information, we obtained with GoSGD in this framework, a time of 0.25 s per SGD iteration for a probability of exchange equal to 0.01. For all the experiments p represents the frequency/probability of communication per batch for one worker. In order to get a fair comparison the methods are always compared at equal frequency/probability of exchange.

5.1. Training speed and generalization performances

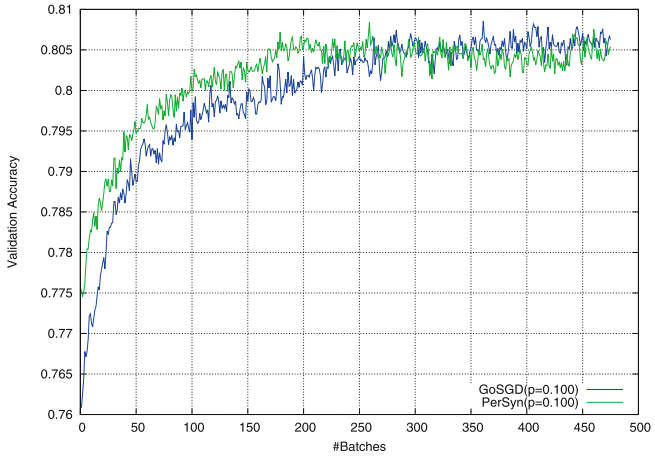
In the following experiments we study the behaviour of the distributed algorithms during training on CIFAR-10. We use the same data augmentation strategy as in [9]. During training, the learning rate is set to 0.1 and the weight decay is set to 10^{-4}. We used eight workers.

We show on Fig. 1 the evolution of the training loss for both PerSyn and GoSGD for different values of the frequency/probability of exchange p. As we can see, PerSyn seems to be slightly faster in terms of convergence speed (as measured by the number of iterations required to reach a specific loss value). However, remember that even not taking into account synchronization issues, PerSyn requires double the amount of message of GoSGD for the same frequency/probability since workers have to wait for the master to answer.

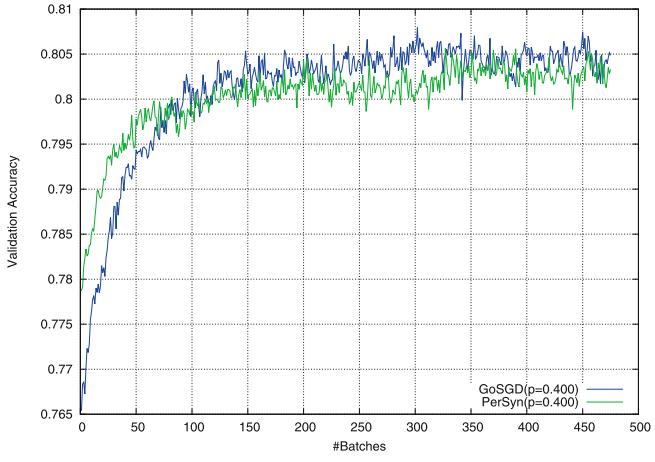
Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002



(a) $p = 0.01$



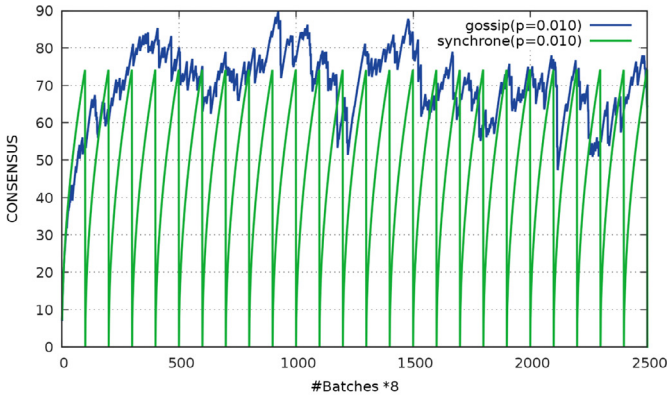
(b) $p = 0.1$



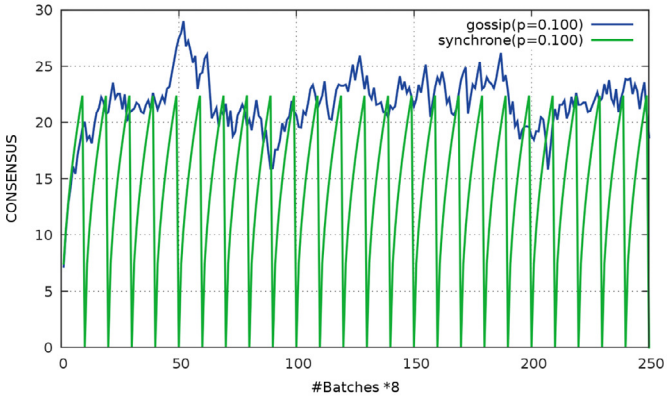
(c) $p = 0.4$

Fig. 3. Validation accuracy evolution for PerSyn and GoSGD for different frequency/probability of exchange p.

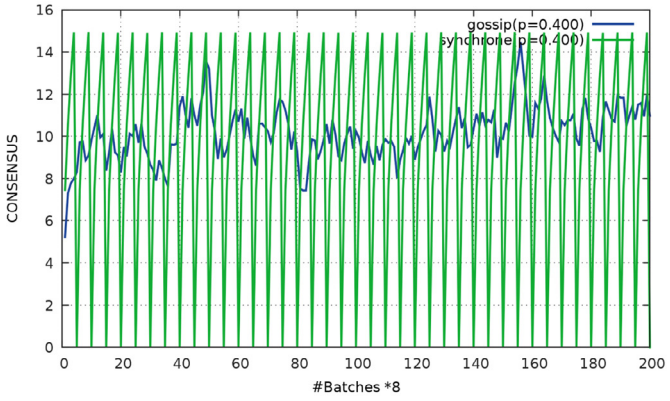
Similarly, we compare the training loss of GoSGD with EASGD on Fig. 2 using a real world clock. As we can see, GoSGD is significantly faster than EASGD, which can be explain by the non blocking updates of GoSGD as well as by the fact that GoSGD requires less messages to keep the same consensus error. finally, we show



(a) $p = 0.01$



(b) $p = 0.1$



(c) $p = 0.4$

Fig. 4. Evolution of the consensus error $\varepsilon(t)$ against time for different communication frequency/probability p for both GoSGD and PerSyn.

on Fig. 3 the evolution of the validation accuracy for both PerSyn and GoSGD for different values of the frequency/probability of communication p. For low frequency $p = 0.01$, despite PerSyn being faster in terms on training loss decrease, both PerSyn and GoSGD models obtain equivalent validation accuracy. However, GoSGD is expected to be at least twice as fast in term of real world clock since it uses half the number of messages for the same rate p.

At higher exchange rate $p = 0.4$, we can see that GoSGD obtains better validation accuracy than PerSyn, despite having higher training loss, which means that GoSGD is in practice more robust to overfitting. This can be explained by the randomized nature of

Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002

the gossip exchanges, which perform some kind of stochastic exploration of the parameter space, similarly to what is done by [26].

Remark that although the batch size is equal for each worker, the implicit global batch size of the algorithms are not equivalent for all curves. Indeed, the single thread run has a batch size of 16 while each of the 8 workers of the distributed runs has a batch size of 16. Therefore the distributed runs have an equivalent batch size of $16 \times 8 = 128$ as calculated in Section 3. These batch sizes are chosen so has to have equal hardware requirements per available machine, which we believe is the most fair and practical use case. In Section 2, we recall that having a bigger batch size leads to a much more accurate gradient updates which may significantly speed up the convergence and this is indeed what we observe in Fig. 1.

5.2. Consensus with random updates

In this experiment, we assess the ability of GoSGD and PerSyn to keep consensus. We consider a worst-case scenario where the local updates are not correlated and as such, we replace the gradient term by a random variable sampled from $\mathcal{N}(0, 1)$, independently and identically distributed on each worker. We show on Fig. 4 the following consensus error for different values of the frequency/probability of exchange p:

$$\varepsilon(t) = \sum_{m=1}^{M} \left\| x_m^{(t)} - \tilde{x}^{(t)} \right\|^2$$

As we can see, at high frequency/probability of exchange, GoSGD and PerSyn are equivalent on average. For low frequency/probability $p = 0.01$, the Gossip strategy has a consensus error in the range of the higher values of PerSyn, as both share the same magnitude. The main difference between the 2 strategies is that PerSyn exhibits the expected periodicity in its behavior which leads to big variation of the consensus error, while GoSGD seems to have much less variation. Overall, GoSGD is able to maintain consensus properties comparable to synchronous algorithms despite its random nature.

6. Conclusion

In this paper, we discussed the distributed computing strategies for training deep convolutional neural networks. We show that the classical stochastic gradient descent can benefit from such distribution since its accuracy depends on the size of the sample batch used at each iteration. We show that distributed SGD involve 2 key operations, namely computation and communication. We also show that there is a crucial trade-off between low communication costs and sufficient communication to ensure that all workers contribute to the same objective function.

We develop the original distributed SGD algorithm into a novel framework that allows to design update strategies that precisely control the communication costs. From this framework, we propose 2 new algorithms PerSyn and GoSGD, the later being based on Randomized Gossip protocols.

In the experiments, we show that both PerSyn and GoSGD are able to work under communication rates as low as 0.01 message/update which renders the communication costs almost negligible. We also show that GoSGD is faster than EASGD [9]. Furthermore, we show that the random communications of GoSGD allows for better generalization capabilities when compared to non-random communications.

Appendix A. Estimator variance

The error introduced by the Monte Carlo estimator of the gradient is proportional to the batch size.

Proof. The relation between the approximation error and the size of the sample batch is given by:

$$\begin{aligned}\mathbb{E}||\nabla L(x) - \widehat{\nabla} L(x)||_2^2 &= \mathbb{E}\left(\sum_{i=1}^{dim(x)} |\nabla L(x)_i - \widehat{\nabla} L(x)_i|^2 \right) \\ &= \sum_{i=1}^{dim(x)} \mathbb{E}|\nabla L(x)_i - \widehat{\nabla} L(x)_i|^2\end{aligned}$$

$\widehat{\nabla} L(x)_i$ being an unbiased estimator of $\nabla L(x)_i$, $\mathbb{E}\left(\widehat{\nabla} L(x)_i\right) = \nabla L(x)_i$ and $\mathbb{E}|\nabla L(x)_i - \widehat{\nabla} L(x)_i|^2 = \mathbb{V}ar(\widehat{\nabla} L(x)_i)$. For Monte-Carlo estimators we have that $\mathbb{V}ar(\widehat{\nabla} L(x)_i) == \mathbb{V}ar(\frac{1}{N}\sum_{k=1}^{n} \nabla \ell(x, Y_k)_i) = \frac{1}{N}\mathbb{V}ar(\nabla \ell(x, Y)_i)$, the Y_k being *iid.*

$$\begin{aligned}\mathbb{E}||\nabla L(x) - \widehat{\nabla} L(x)||_2^2 &= \sum_{i=1}^{dim(x)} \frac{1}{N}\mathbb{V}ar(\nabla \ell(x, Y)_i) \\ &= \frac{1}{N} tr(\mathbb{C}ov_Y[\nabla \ell(x, Y)])\end{aligned}$$

□

Appendix B. Alternate gradient

The update rules of GoSGD are in expectation equivalent to the empirical risk and the consensus loss parts of the gradient of the consensus augmented distributed problem.

For the empirical risk part, since at each time step t one of the workers performs a local stochastic gradient descent update with probability $1/M$, it is in expectation equivalent to performing the gradient descent on the empirical risk part with a learning rate divided by M.

For the consensus part, let us recall that the update for a receiving worker is:

$$x_r^{(t+1)} = \frac{w_r}{w_s + w_r} x_r^{(t)} + \frac{w_s}{w_s + w_r} x_s^{(t+)}$$

This update is occurring with probability $\frac{p}{M(M-1)}$ for all pairs (x_s, x_r) of workers. First we demonstrate the following lemma:

Lemma 1. $\mathbb{E}\left[\frac{w_r}{w_r + w_s}\right] = \frac{1}{2}$ *for all possible pairs* (w_r, w_s).

Proof. The weights update can be written with a random communication matrix $A^{(t)}$ with the following definition:

$$A^{(t)} = \begin{cases} I, \text{ with probability } p \\ I - \frac{1}{2} e_s e_s^\top + \frac{1}{2} e_r e_s^\top, \text{ with probability } 1 - p \end{cases}$$

The sequence of $A^{(t)}$ are i.i.d. and we note $\mathbb{E}\left[A^{(t)}\right] = A$:

$$\begin{aligned}A &= pI + (1-p) \sum_{s, r \neq s} \frac{1}{M(M-1)} \left(I - \frac{1}{2} e_s e_s^\top + \frac{1}{2} e_r e_s^\top \right) \\ &= \left(1 - \frac{1-p}{2M}\right) I + \frac{1-p}{2M(M-1)} \mathbb{1}\mathbb{1}^\top\end{aligned}$$

Using A and denoting $w^{(t)}$ the vector concatenating all weights at time t, we have the following recursion:

$$\begin{aligned}\mathbb{E}\left[w^{(t+1)} | w^{(t)}\right] &= \mathbb{E}\left[A^{(t)}\right] w^{(t)} \\ &= A w^{(t)}\end{aligned}$$

Since all weights are initialized to 1, unrolling the recursion leads to:

$$\mathbb{E}\left[w^{(t+1)}\right] = A^t \mathbb{1}$$

Since $\mathbb{1}$ is a right eigenvector of A (associated with eigenvalue λ), we have:

$$\mathbb{E}\left[w^{(t+1)}\right] = \lambda^t \mathbb{1}$$

Which means that all weights are equal in expectation. □

Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002

This lemma leads to the following expected update step:

$$\mathbb{E}\left[x_r^{(t+1)} - x_r^{(t+\frac{1}{2})}\right] = \frac{p}{2M(M-1)}\left(x_s^{(t)} - x_r^{(t+\frac{1}{2})}\right)$$

Which corresponds in expectation to a gradient descent performed on the consensus part with a learning rate of $\frac{p}{2M(M-1)}$.

References

[1] K. Fukushima, Neocognitron: a self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position, Biol. Cybern. 36 (4) (1980) 193–202.
[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[3] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Proceedings of the Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet Large scale visual recognition challenge, Int. J. Comput. Vis. (IJCV) 115 (3) (2015) 211–252.
[5] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C.L. Zitnick, P. Dollr, Microsoft coco: common objects in context, arxiv (2015).
[6] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of the COMPSTAT'2010, Springer, 2010, pp. 177–186.
[7] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[8] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, (2014). arXiv:1409.1556.
[9] S. Zhang, A.E. Choromanska, Y. LeCun, Deep learning with elastic averaging SGD, in: Proceedings of the Advances in Neural Information Processing Systems 28, 2015, pp. 685–693.
[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q.V. Le, et al., Large scale distributed deep networks, in: Proceedings of the Advances in Neural Information Processing Systems 25, 2012, pp. 1223–1231.
[11] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Randomized gossip algorithms, IEEE Trans. Inf. Theory 52 (2006).
[12] I. Colin, A. Bellet, J. Salmon, S. Clémençon, Gossip dual averaging for decentralized optimization of pairwise functions, in: M.F. Balcan, K.Q. Weinberger (Eds.), Proceedings of the Thirty-Third International Conference on Machine Learning, Proceedings of Machine Learning Research, New York, NY, USA, 48, 2016, pp. 1388–1396.
[13] J. Fellus, D. Picard, P.-H. Gosselin, Asynchronous gossip principal components analysis, Neurocomputing 169 (2015) 262–271.
[14] G. Di Fatta, F. Blasa, S. Cafiero, G. Fortino, Epidemic k-means clustering, in: Proceedings of the IEEE Eleventh International Conference on Data Mining Workshops (ICDMW), IEEE, 2011, pp. 151–158.
[15] J. Fellus, D. Picard, P.-H. Gosselin, Decentralized k-means using randomized gossip protocols for clustering large datasets, in: Proceedings of the IEEE Thirteenth International Conference on Data Mining Workshops, IEEE, 2013, pp. 599–606.
[16] V. Vapnik, The Nature of Statistical Learning Theory, Springer, 1995.
[17] D.G. Luenberger, Y. Ye, Linear and Nonlinear Programming, Springer, 2008.
[18] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Proceedings of the International Conference on Learning Representations, 2015.
[19] G. Hinton, Overview of mini-batch gradient descent, in: Neural Networks for Machine Learning, Lecture 6a, 2013 https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.
[20] G. Huang, Z. Liu, L. van der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.
[21] H. Ma, F. Mao, G.W. Taylor, Theano-MPI: a theano-based distributed training framework, in: Proceedings of the European Conference on Parallel Processing, Springer, 2016, pp. 800–813.
[22] A. Choromanska, M. Henaff, M. Mathieu, G.B. Arous, Y. LeCun, The loss surfaces of multilayer networks, in: G. Lebanon, S.V.N. Vishwanathan (Eds.), Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research, San Diego, CA, USA, 38, 2015, pp. 192–204.
[23] D. Kempe, A. Dobra, J. Gehrke, Gossip-based computation of aggregate information, in: Proceedings of the Forty-Fourth Annual IEEE Symposium on Foundations of Computer Science, IEEE, 2003, pp. 482–491.
[24] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends Mach. Learn. 3 (1) (2011) 1–122.
[25] A. Krizhevsky, Learning Multiple Layers of Features From Tiny Images, Computer Science Department University of Toronto, 2009 Ph.D. thesis.
[26] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, R. Fergus, Regularization of neural networks using dropconnect, in: Proceedings of the International Conference on Machine Learning, 2013, pp. 1058–1066.
[27] R. Collobert, C. Farabet, K. Kavukcuoglu, S. Chintala, Torch 7, (http://torch.ch/).

Michael Blot received a M.Sc in Mathematical Logic and Computer Science Fundamentals from University Paris Diderot in 2014 and a M.Sc in quantitative finance of IAE of Grenoble in 2012 and he received the Engineering Diploma in applied mathematics from Ensimag in 2012. He is currently a Ph.D. student in the Machine Learning and Information Access of University Pierre et Marie Curie, under the supervision of Matthieu Cord. His research focuses on deep learning for image classification and in particular on training methods of convolutional neural networks.

David Picard received the M.Sc. in Electrical Engineering in 2005, the Ph.D. in Image and Signal Processing in 2008 and the Habilitation in Computer Science in 2017. He joined the ETIS laboratory at ENSEA Graduate School (France) in 2010 as an associate professor. His research interests include computer vision and machine learning, with a focus on kernel methods, deep learning and distributed learning with applications to image indexing and retrieval.

Nicolas Thome is a full professor at Conservatoire Nationnal des Arts et Metiers (Cnam Paris). He received the Ph.D. degree in computer science from the University of Lyon, France in 2007, and has been associate professor at UPMC-Paris 6 from 2008 to 2016. His research interests include machine learning for computer vision, including applications for semantic understanding of multimedia data. He is involved in several French (ANR), European and international (Canada, Singapore, Brazil) research projects. He is being coordinator of an ANR project on weakly supervised learning for image retrieval in 2013–2018.

Matthieu Cord is a full professor at Sorbonne University. He received the Ph.D. degree computer science from the UCP, France, before working as postdoc at KU Leuven, Belgium. His research interests include computer vision, deep learning and artificial intelligence. He developed several interactive learning-based approaches for CBIR and many models for pattern recognition using deep architectures. Recently, he focused on multimodal (vison and language) understanding. M. Cord is (co-)author of more than 100 international, peer-reviewed publications among including two edited books. He is involved in several French, European and international research projects. In 2009, he was nominated to the prestigious IUF (French Research Institute).

Please cite this article as: M. Blot, D. Picard and N. Thome et al., Distributed optimization for deep learning with gossip exchange, Neurocomputing, https://doi.org/10.1016/j.neucom.2018.11.002