Scalable Machine Learning and Deep Learning - Review Questions 6

Deadline: December 15, 2019

Anna Martignano, Daniele Montesi

May 4, 2020

1. What are model parallelism and data parallelism?

   Both of them represent two approaches to train a single model across multiple devices.
   Model parallelism encompasses the partition of the model into chunks over multiple devices. This approach is not very straightforward to implement since it is strongly dependant on the architecture of the neural network. For example, if we perform the partition horizontally, placing each layer on a different device, we do not improve significantly the computation time since each layer is strongly dependent to the previous one, and the execution still happens in a sequential order.
   To really achieve a parallel execution the solution is to perform the partition vertically, but a lot of cross-device communication is required since each layer requires as an input the output of all the partitions.
   For these reasons, data parallel approach is preferred. It consists of the replication of the whole model on every device and all the replicas are trained simultaneously. The strong assumption to implement data parallelism is that the model should be able to fit in a single device.
   The data parallelization steps are the following:

   - Computations of the loss function gradients using a mini-batch on each GPU.
   - Computation of the mean gradients by inter-GPU communication.
   - Model update.

   The implementation challenge for data parallelism is to establish how and when perform the parameters synchronization among all the model instances instantiated on the different devices.

---

2. When training a model across multiple servers, what distribution strategies can you use? How do you choose which one to use?

   The selection of the distribution strategy used to train a model of course is strongly dependent to the requirement of the systems. In the following lines is provided an overview of the implementation technique which can be used.

   When training a model over a distributed network there can be multiple ways to handle distributed training: it is possible distribute either the model (Model parallel) or the data (Data parallel).

   - Model Parallel are the techniques regarding the model distribution over multiple nodes. The problems with these approaches consists in the difficulties to reach an high speed up due to the network overhead and because of the sequential nature of the models. When talking about Neural Network, going to distribute the traditional Feed-Forward NN results bad. However, good results can be achieved then distributing a RNN or a CNN, which are special types of layer that can be easily parallelized.
   - Data Parallel is usually the chosen alternative when dealing with traditional Neural Networks or models that cannot be easily distributed.

When talking about the System Architecture, instead, there are two strategies to aggregate the gradients.

- Centralized System with the use of a Parameter Server. The server is a shared storing of the parameters: it gets all the parameters from the nodes and averages them to get the parameters with contribution of every node. This architecture suffers of unique point of failure.
- Decentralized System: every node shares its parameters between each other and a computation is performed locally to determine only a single set of parameters. This operation is called Reduce function. The reduce function is computed either on all the nodes (waste of resources) or in an elected master node (unique point of failure).
  1. In AllReduce architecture, every node computes the reduce function and the number of messages passed is limited asymptotically by a factor $O(N \times (m-1))$.
  2. Ring-AllReduce is improving this: nodes are logically distributed in a ring and communicates between each other only a "Fraction" of the total data. Sequentially, the data is passed to the other nodes. This usually results in lower overhead costs (the number of messages passed is limited asymptotically by a factor $O(\frac{N}{m} \times (m-1))$

In general, when performing a computation (i.e. training a model) we should minimize the number of messages, hence if we want Decentralized system we should opt for RingAllReduce, otherwise for the Parameter Server centralized architecture, which usually is the best choice if the parameters to update are few.

While implementing the system architecture, it must be considered as well how to synchronize the parameters among different replicas. Synchronization may happen in a synchronous fashion: after every iteration workers exchange communication with each others. From one side, this approach has the advantage to simplify the reasoning regarding convergence but the slowest worker negatively impacts the speed of the entire system. In an asynchronous system instead, workers may train on stale parameters by renouncing to accurate mathematically reason abot the convergence.

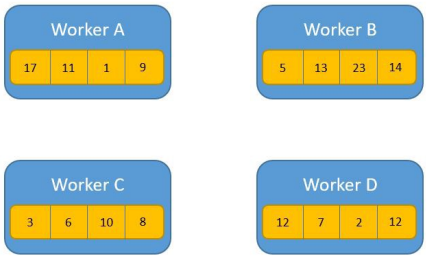
---

3. Briefly explain gradient quantization and gradient sparsification.

   - **Gradient Sparsification:** The main idea of Gradient Sparsification is to randomly drop out coordinates of the stochastic gradient vectors and amplify the remaining coordinates appropriately. The goal is to ensure the sparsified gradient to be unbiased. However, it must be reached an optimal sparsification and that requires computation and reaching a theoretical guarantees for sparseness. In the literature many approaches are available to achieve this.
   - **Gradient Quantization:** The main idea in Gradient Quantization is to reduce the communication overhead by trasmitting "quantized" information of the gradient coefficients. In this approach, gradient is decomposed into its norm and the normalized stochastic gradient. The successive transformation depends on the type of quantized, for instance, in the hierarchical quantizer, the norm is quantized using a scalar quantizer, the high-dimensional normalized stochastic gradient is decomposed further into a set of equal-length unitary vectors and a hinge vector.

---

4. Use the following picture and show step-by-step how the ring-allreduce works to compute the sum of all elements?



Ring-AllReduce is a decentralized system architecture which enables the parameters synchronization among different model instances located on multiple devices. In decentralized architectures, the model parameters are mirrored across all the workers, and the Ring-AllReduce technique determines how the parameter updates are exchanged among the workers directly through an allreduce operation.
As we can see from the picture, each instance's array of data is divided into $m = 4$ chunks since we have 4 instances of the model. Ring-AllReduce comprehends two phases:

- Share-Reduce phase - each instance p sends data to the process $(p + 1) \mod m$; when a process receives the data from the previous process, it applies the reduce operation and send again the results of the reduce to the next process in the ring until each process holds the complete reduction of chunk i.
- Share-Only phase - this step is performed to consolidate the result obtained by each chunk by simply sharing data in a ring-like fashion but without perform any reduce operation.

The complexity of this method is $O(\frac{N}{m} \times (m-1))$ because in the share-reduce step each process sends $\frac{N}{m}$ elements and perform this operation $(m-1)$ times.

If we assume that the reduce operation is a sum we obtain the following results for each iteration:

Iteration	**Worker A**				**Worker B**				**Worker C**				**Worker D**			
0	17	11	1	9	5	13	23	14	3	6	10	8	12	7	2	12
1		11	1	**21**	**22**		23	14	3	**19**		8	12	7	**12**	
2		11	**13**				23	**35**	**25**			8	12	**26**		
3		**37**					**36**					**43**	**37**			
4	37	37				37	36				36	43	37			43
5	37	37		43	37	37	36			37	36	43	37		36	43
6	37	37	36	43	37	37	36	43	37	37	36	43	37	37	36	43