I-RIM Conference 2019
October 18 - 20, Rome, Italy
ISBN: 9788894580501
DOI: 10.5281/zenodo.4785833

Training Neural Networks on Resource-Constrained Devices

Franco Maria Nardini
ISTI-CNR, Pisa, Italy
francomaria.nardini@isti.cnr.it

Lorenzo Valerio
IIT-CNR, Pisa, Italy
lorenzo.valerio@iit.cnr.it

Andrea Passarella
IIT-CNR, Pisa, Italy
andrea.passarella@iit.cnr.it

Raffaele Perego
ISTI-CNR, Pisa, Italy
raffaele.perego@isti.cnr.it

***Abstract*—The digital transformation we are experiencing in recent years is cross-cutting to all sectors of the society. In the industrial scenario, this transformation is leading towards the fourth industrial revolution characterized by i) large amounts of data collected and ii) decentralization of computational resources along the production line. In this context the use of artificial intelligence (AI) is often subordinated to the adoption of distributed solutions characterized by the use of limited hardware capacity. In this paper, we describe a new framework for learning neural networks on devices with limited resources. A first experimentation on MNIST datasets confirms the validity of the approach that allows to effectively reduce the size of the network during training without significant losses of its accuracy.**

***Index Terms*—Neural networks, resource-constrained devices**

I. Introduction

Nowadays, global companies are facing an important transition towards the Industry 4.0 paradigm. In this paradigm a central role in the development and the optimization of the production process is given to data management and its analysis in order to extract useful knowledge for improving the industrial process. This transformation will be made possible by i) the increasing availability of data produced at all the levels of the production process, together with ii) the presence of more powerful robots (i.e. with improved sensing and computational computational) distributed over the entire production line. As an example, consider two typical case studies in Industry 4.0: i) process optimization, to improve the efficiency of a production chain adapting the behaviour of the robots according to newly obtained knowledge, ii) predictive maintenance, to optimize the lifetime of an industrial plant using the data of the plant itself. However, given the exponential growth of data produced and collected at the edge of the network and specifically within the industrial premises, it is expected that the capacity of the communication and computing infrastructures (despite the advent of 5G and the evolution of data centers) will not deal with the "data tsunami" expected in the coming years, making the paradigm based solely on cloud-based solutions (i.e. Cloud Robotics [1]) no longer sustainable in the medium-long run [2]. Given these considerations, there is an increasing need to combine centralized AI based on cloud systems with decentralized AI solutions that are physically located directly on (or close to) the devices (e.g. industrial robots) that are involved in the production processes. For example, a viable solution would be the adoption of paradigms like edge or fog computing [3]. According to this paradigms in which part of the knowledge extraction is being shifted directly on (or close to) the end devices (i.e. robots), becomes fundamental to develop AI techniques that are suitable for the limited resources at their disposal. This is particularly relevant to scenarios characterised by a intelligent (industrial) machines (IMs) equipped with a range of small devices with data processing capabilities. Novel, efficient AI solutions are suitable for the Industry 4.0 scenario that robots can use to train efficiently local accurate models, i.e., using the data they own or sense.

While solutions already exist in using, e.g., Deep Neural Networks on resource constrained devices *after* they are trained on remote data centres, limited results are available to enable efficient *training* of DNNs on such devices, based on local data. In this paper, we focus on machine learning solutions suitable for saving resources during learning phase. To this end, we propose to exploit compression techniques for deep neural networks during the learning process so to allow a significant reduction of the size of the neural network while training it. Our purpose is twofold. On the one hand we want to reduce the computational burden of executing the trained final model (i.e. a smaller neural network is computationally less expensive that a bigger one) and, on the other hand, we want also to reduce the amount of resources used at training time, i.e., memory. Note that here we focus on the memory because of it is well known that, in terms of power consumption, accessing the memory is in the range of one order of magnitude more expensive that performing arithmetic operations [4].

Compression techniques (pruning) of neural networks follow two main research lines. While the first line focuses on removing some of the neurons of the network, the second line works by removing unnecessary weights. Most of the work so far focuses mainly on the second approach. The main contributions in this line are based on: i) the factorization of the weights matrix, ii) limiting the magnitude and/or the representation of the values of the weights and iii) using *distillation* methods. The main criticism moved to the works belonging to this group is that the use of compression is typically employed after training the network. A recent and

This work is partially supported by three projects: i) EU H2020 AUTOWARE, GA # 723909, ii) EU H2020 BigDataGrapes, GA # 780751, iii) MIUR PON OK-INSAID, ARS01_00917.

alternative research line proposes to integrate compression approaches during the learning phase. These approaches aim at learning the distribution of the neurons' activation during the training. Such information are used to turn on (off) the neurons that are more (less) useful.

II. Hard Pruning of Neural Networks

In this work we take inspiration from a state-of the art approach where the network pruning takes place by learning the activation/deactivation distribution of the network neurons [5]. The proposed approach trains a Deep Neural Network minimizing a regularised loss function. The loss function is composed by two terms that are a function of the parameters of the network $\boldsymbol{\theta}$: i) the accuracy of the network and ii) a L_0-norm regularizer $\|\boldsymbol{\theta}\|_0$ through which we seek the minimum set of parameters that maximizes the accuracy of the network. In details,

$$R(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} L(h(x_i;\boldsymbol{\theta}), y_i) + \lambda\|\boldsymbol{\theta}\|_0 \qquad (1)$$

where, h represents a neural network whose parameters are identified by $\boldsymbol{\theta}$. Moreover, N is the size of the dataset composed by independent and identically distributed pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$. Equation 1 is minimized in the number of parameters $\boldsymbol{\theta}$.

Since the learning of the network is done via *backpropagation*, it is not possible to directly use the loss function in Equation 1 due to the non-derivability of the L_0 regularizer. This problem has been overcome through a probabilistic approach, i.e., by approximating the distribution of active/deactive neurons through a continuous and derivable distribution, namely the *Hard Concrete Distribution* [5]. As a result, it is possible to jointly minimize the prediction error and the number of active neurons, epoch by epoch. However, the reduction introduced by this pruning solution is "virtual" and commonly named *soft pruning*. In fact, although the number of active neurons decreases, due to the random extraction process, the active neurons are always different epoch by epoch. From the point of view of the use of resources, this does not guarantee any memory savings since the size of the network is constant, i.e., neurons are simply turned on and off epoch by epoch.

The problem above can be solved by introducing *hard pruning*. The idea is that, during learning, if a neuron is turned off it cannot be turned on again. Training with hard pruning has the effect of clearing, at training time, some of the weights and consequently, parts of the network. Differently from soft pruning, this mechanism allows an effective reduction of the memory occupation of the network. In fact, to effectively hard prune the network, we remove the rows of the weights matrices whose neurons are zeroed to obtain matrices with a lower number of elements. However, since with hard pruning neurons are turned off definitively during training, we must pay attention in selecting those neurons to not interfere too much the training process degrading the prediction capabilities of the network. To this end, we adopt the following probabilistic approach: during each epoch we train the network using soft-pruning and, in the meanwhile, we compute the empirical distribution of the neurons' activation. In this way we obtain, for each neuron, its probability of "being active". From the probabilities obtained, we create a binary mask that is applied to the network during the following epochs. This mask, updated epoch by epoch, helps to shut down the nodes of the network that were previously switched on while maintaining the overall state of hard pruning during training. The gradient descent, under the hard pruning settings becomes very sensitive because from one epoch to the next, part of the network disappears. Therefore, it becomes important to drive the optimization process properly. In this paper, instead of tuning the learning rate, we dynamically resize the mini-batches according to the accumulated variance of the gradients during each epochs. The idea is to adapt the batch size inversely proportional to the gradients' variance in order to incrementally reduce it as long as the network learns. As a byproduct, as long as the learning goes on the memory allocated to the mini batches is reduced to the actual amount used. This further reduces the total memory footprint of the entire learning process.

III. Preliminary Results and Conclusions

We evaluated the effectiveness of our hard pruning mechanism described in Section II on a multi-label classification task: the recognition of a handwritten number from an image. The task asks to assign to each image of the dataset a class that identifies the handwritten number in the image itself. To this regard, we used the well-known MNIST dataset. We trained a Multilayer Perceptron (MLP) for 500 epochs with two hidden layers of 300 and 100 nodes, respectively. The network consists of 266.610 parameters for a total of about 1 MiB of memory used. The results of the network trained with hard pruning are then compared with those of the MLP trained using soft pruning.

The preliminary results obtained show that the application of hard pruning reduces the size of the network during training of up to 50% with a performance degradation of accuracy of less than 0.2% compared to the equivalent network trained using soft-pruning.

As future work we intend to refine the proposed method and to conduct a more comprehensive evaluation on its performance.

References

[1] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, April 2015.

[2] I. Cisco Systems, "Cisco Global Cloud Index: Forecast and Methodology, 20162021," *White Pap.*, p. 46, 2018.

[3] M. Conti, A. Passarella, and S. K. Das, "The Internet of People (IoP): A new wave in pervasive mobile computing," *Pervasive Mob. Comput.*, vol. 41, pp. 1–27, 2017.

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015.

[5] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv:1712.01312*, 2017.