Distributed neural networks for Internet of Things: the Big-Little approach

Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Steven Bohez, Pieter Simoens, Piet Demeester, and Bart Dhoedt

Ghent University – iMinds,
Department of Information Technology
Gaston Crommenlaan 8/201
9050 Gent, Belgium
elias.deconinck@intec.ugent.be

Abstract. Nowadays deep neural networks are widely used to accurately classify input data. An interesting application area is the Internet of Things (IoT), where a massive amount of sensor data has to be classified. The processing power of the cloud is attractive, however the variable latency imposes a major drawback for neural networks. In order to exploit the apparent trade-off between utilizing the available though limited embedded computing power of the IoT devices at high speed/stable latency and the seemingly unlimited computing power of Cloud computing at the cost of higher and variable latency, we propose a Big-Little architecture for deep neural networks. A small neural network trained to a subset of prioritized output classes can be used to calculate an output on the embedded devices, while a more specific classification can be calculated in the Cloud only when required. We show the applicability of this concept in the IoT domain by evaluating our approach for state of the art neural network classification problems on popular embedded devices such as the Raspberry Pi and Intel Edison.

Key words: Deep Neural Networks, Distributed Intelligence, Internet of Things

1 Introduction

Currently, the Internet of Things (IoT) is a popular paradigm that envisions a world in which all kinds of physical objects or "things" get connected to the Internet, being able to interact with each other and cooperate to reach common goals [1]. This goes beyond machine-to-machine communications (M2M), as it covers not only communication protocols, but also the application domains and the services running on top of these connected things. By providing easy access to a myriad of devices such as sensors, surveillance cameras, home appliances, cars, actuators etc., the IoT will enable a new range of applications and use cases in the field of domotics, assisted living, logistics, smart environments, manufacturing, and many more.

In order to create truly smart applications for the IoT, massive amounts of data coming from all connected things will have to be processed and analysed into actionable and contextualized information [7]. Currently, Cloud computing is most often the natural choice to perform this data processing, benefiting from the huge compute power and scalability of current Cloud infrastructure [14]. However, the Cloud is not a silver bullet solution, as a network connection to a Cloud datacenter often suffers from high and variable latency, as well as a limited upload bandwidth [2]. Moreover, extensive Cloud processing incurs considerable cost. Therefore, our goal is to first use local compute power in the various embedded devices and gateways for data processing, before turning to the Cloud. This addresses the problem associated with network connectivity (latency and bandwidth) while also reducing operation cost.

A very important processing step in IoT applications is classification, i.e., determining the system "state" and its subsequent actions based on (sequences of) sensory data. A promising state-of-the-art technique in this field is the use of a Deep Neural Network (DNN), which offers a biologically inspired trainable architecture that can learn various invariant features [17]. As a neural network is trained by iteratively evaluating input samples and updating the neural network weights, one often relies on efficient GPU implementations that can exploit the parallelism and speedup neural network evaluation. However, the size of the neural network that can be loaded onto an IoT device will be limited by both its available CPU power and internal memory.

The contributions presented in the current paper is a Big-Little architecture for neural networks, that is tailored to the specific characteristics of IoT environments. The idea is to take a large trained neural network for a certain classification problem, and from that distil a smaller neural network that only classifies a well chosen subset of the output space. The little neural network is suited to be executed locally on the embedded devices, whether the input is also sent to the Cloud, for evaluation with the big neural network, is upto the application specific demands.

The rationale for this idea is the following. Consider a smart home environment that is monitored by a large number of sensors and some actuators. Possible actions the smart home can perform are to trigger an alarm, start the heating system, detect fire or leaks, etc. In these cases, some situations are critical to detect fast, in order to take direct action. For example trigger an alarm when a person enters the house, or close off the water supply when a leak is detected. One can clearly distinguish between critical situations, in which you want fast and reliable response, versus non-critical ones, as well as the granularity of the response (a person versus which specific person). By carefully selecting the critical outputs and the granularity of the outputs, we can craft a little local neural network that offers fast response. This local neural network also acts as a filter to limit the number of inputs sent to the big neural network in the cloud. For example, only when a person is detected and one requires a more fine grained classification of the person, the input is evaluated in the Cloud.

The remainder of this paper is organized as follows. Section 2 presents related work in scope of distributed neural networks. Section 3 explains the proposed Big-Little architecture for neural networks in the scope of IoT. In Section 4 we show some preliminary results that validate our idea using a frequently used neural network evaluation dataset. Section 5 summarizes our conclusions and presents plans for future work.

2 Related work

In a highly distributed IoT environment one could try to speed up neural network evaluation by distributing parts of the neural network among all the available devices. However, related work in distributing neural networks shows that the communication overhead quickly becomes the limiting factor [6], hence this approach is mainly used to speed up the training phase on a cluster of nodes connected through a high speed network. Krizhevsky et al. [11] showed how a larger neural network can be trained by spreading the net across two GPUs. This way the communication overhead remains limited, as the GPUs are able to read and write to each other's memory directly. In [9], the authors show that scaling up further to 8 GPUs can lead to a speed up factor of 6.16. Dean et al. [8] presented the DistBelief framework for parallel distributed training of deep neural networks. By adopting new training algorithms they can distribute the training procedure on a large number of CPU nodes, for example achieving a speed up of more than 12x using 81 machines.

As these methods all focus on the training phase and require high end server infrastructure, these are not applicable for speeding up small neural networks in an IoT use case. One approach to optimize distributed neural network execution with parts distributed across embedded devices is simplifying multi-class classification problems to one-vs-all (OVA) or one-vs-one classifiers (OVO). These methods are a straightforward and often used to construct a multi-class classifier using binary-class classification [15]. One multi-class classification is split into a set of binary-class classifications for each class and later combine them to the original multi-class classifier.

An other optimization approach in which one part is deployed on an embedded device, and a second part is running in the Cloud, uses a cascade of neural network layers as depicted in [13]. The presented solution exists in augmenting the structure of the neural network to obtain intermediate evaluation output. Then, the evaluation with the remaining neural network layers is pre-empted if the quality of the intermediate output exceeds a given threshold. By deploying only first layers of the net on an embedded device, calls to the Cloud can be limited to the input samples that do not yet result in a good output after these first layers. In contrast, this paper proposes a similar approach, but instead of training intermediate layers that classify all outputs with a lower accuracy, we introduce a smaller neural network that is trained to only a subset of the outputs.

3 Big-Little neural network architecture

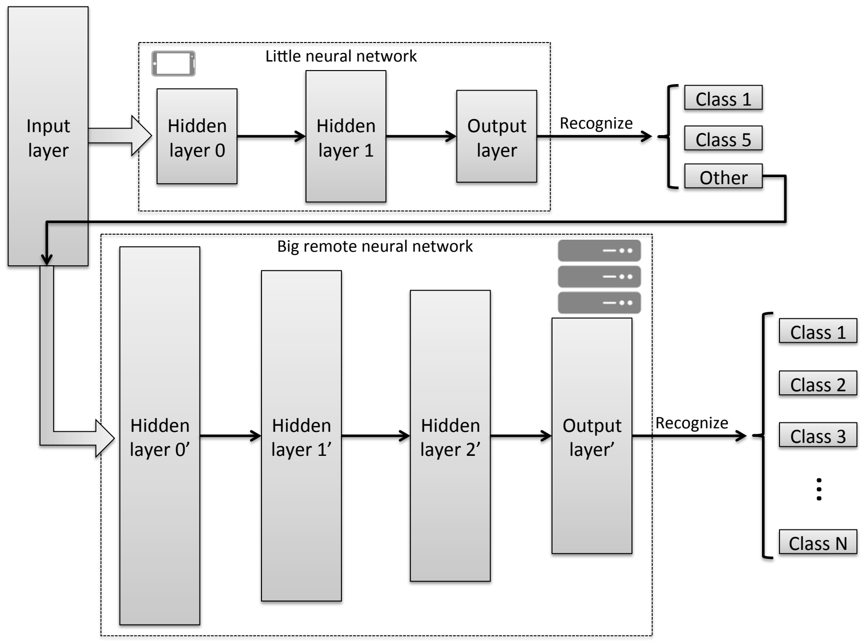

Fig. 1. Architecture of Big-Little neural network: the little neural network only classifies a subset of the output classes, and can be executed locally with limited CPU power. When the little neural network cannot classify the input sample, a big neural network running in the Cloud can be queried.

A typical feed forward neural network is composed of an input layer, one or more hidden layers and one output layer. The output layer has one output for each classification class, resulting in a value between 0 and 1, depicting the probability that the input can be classified as such. In order to process as much as possible locally on embedded IoT devices, and limiting communication to the Cloud, we designed a little neural network that can be processed with limited CPU power, which classifies only a subset of the output classes. This results in a Big-Little neural network architecture as depicted in Figure 1.

The hypothesis of this paper is that by limiting the number of output classes, we can also limit the size and amount of hidden layers of the neural network, while maintaining the desired classification accuracy. By crafting the little neural network to classify high priority classes only, a local response is obtained very fast in these critical cases, while retaining the availability of the complete classification set by relaying to the cloud, that executes the big neural network.

Whether the input is sent to the cloud is upto the application developer to decide. Directly sending it to the cloud allows for a better classification but usages more bandwidth, while waiting for completion of the little network takes some time.

Because the big and little neural network only share the input layer, these two networks can be trained independently from each other using state-of-the-art training techniques in an offline training phase.

4 Evaluation

To train and execute our neural networks, the popular MNIST dataset [12] was used. This dataset consists of a training, validation and test set with a total of 70000 examples evenly distributed across the classification classes. We made use of the Theano [3], [4] python module, which is compatible with GPU/CPU and many computer architectures. In our experiments we use multilayer perceptrons (MLP) with one or more hidden layers to easily increase the number of calculations by increasing the number of neurons in each hidden layer. Starting with the state of the art neural network of Ciresan et al. [5] as the big network that is executed on the cloud, we have distilled a small network by prioritizing some output classes and aggregating the others into one category, thus limiting the number of output classes. For this latter situation, a neural network structure was created that achieves the same accuracy for the prioritized output classes with a much lower neural network size.

4.1 execution time depends on hardware capabilities and network complexity

To asses device performance a random MLP, with increasing number of weights, is generated and executed with a randomly generated input sample. Since purely evaluating wall-clock time does not require realistic training and input samples. We ran our experiments on multiple devices shown in table 1.

Table 1. Hardware specifications.

name	architecture	CPU	RAM
Raspberry Pi 2	ARM	Cortex-A7 (quad-core @ 900 MHz)	1 GB
Intel Edison	x86	Intel Atom (dual-core @ 500 MHz)	1 GB
Generic server	x86	2x Intel Xeon E5-2650v2 (8-core @ 2.60 GHz)	48 GB

In figure 2 we compare the CPU strength of two embedded devices to a generic server (table 1). The execution time linearly scales with the number of weights in the neural network. From figure 2 we can conclude that the latency, which is the unidirectional delay from one node to another, is more important

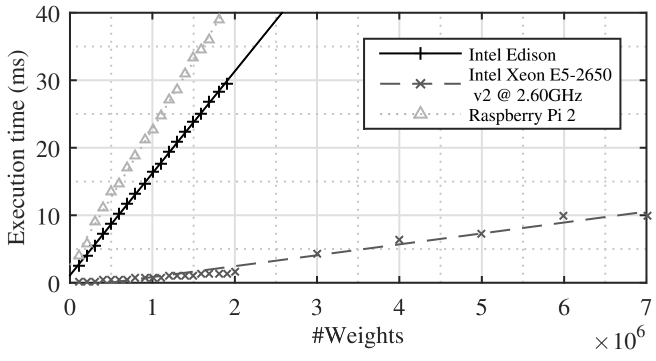

Fig. 2. Limitations of embedded devices (RasberryPi and Intel Edison) for increasing number of weights in hidden layer of a fully connected neural network. Execution time was measured for a single input image. Extra cores do not improve performance.

than the actual calculation. Given we want a maximum response time (RT), which includes the actual calculation and latency, of 40ms and there is a single link latency of 15ms to the server we can run a neural network on a local Raspberry Pi 2 with 2×10^6 weights or a remote neural network with a maximum execution time of max RT $- 2 *$ latency $= 10ms$ which has around 6.5×10^6 weights. In other words the maximum latency between nodes is half of the maximum RT minus the time needed to execute the neural network.

4.2 classification accuracy depends on neural network complexity

The second experiment compares the accuracy between the little one-vs-all neural network to the big multi-class neural network. By increasing the number of

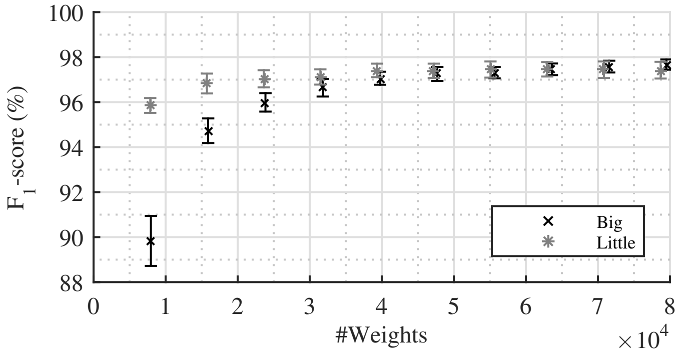

Fig. 3. F_1-score of big and little fully connected neural networks for MNIST classification. The graph shows the F_1-scores for classifying a '5' character on the big network (ID 3 - table 2) and the little network (ID 1 - table 2).

weights in the hidden layer of both networks we can evaluate the difference in reliability. To fairly compare these networks we use the F_1-score of one priority class:

$$F_1\text{-score} = \frac{2 * \text{true positives}}{2 * \text{true positives} + \text{false positives} + \text{false negatives}}$$

This F_1-score is the harmonic mean of precision and recall and is independent of the total number of samples. Comparing the error rate of a multi-class classifier to a binary classifier would yield misleading results because the one-vs-all classifier is trained using a unbalanced sample set. The 'all' class has more samples compared to the 'one' class. The F_1-score of a multi-class neural network can be calculated for each class, but here we only look at the priority class of the little network.

Figure 3 shows a comparison between a big and a little fully connected neural network. A more detailed output is given in table 2 and 3. Neural networks with more output layers need more weights to reach the same accuracy. Our one-vs-all network outperforms the big neural network for a low number of neural network connections and more or less reaches the same quality of larger networks.

Table 2. Execution time of big and little neural networks on the server from table 1. Each number in the architecture represents the number of neurons in that layer, starting with an input layer of $28 * 28$ neurons for MNIST. The last layer represents a binary or multi-class classifier layer.

ID	architecture (number of neurons in each layer)	weights	process time for one sample [ms]
1	784, 1 000, 500, 2	1 250 502	0.98
2	784, 1 000, 500, 10	1 254 510	1.00
3	784, 2 500, 2 000, 1 500, 1 000, 500, 10 [5]	11 972 510	16.42

Table 3. F_1-score comparison of a big (ID 3) and a little (ID 1) neural network for two classification classes (1 and 8) of MNIST. Structure of these networks are shown in table 2. The test error is not applicable for little networks with a unbalanced input sample set.

			priority class [%]			
ID	priority class	test error for best validation [%]	recall	specificity	precision	F_1-score
1	1	NA	98.94	99.92	99.38	99.16
	8	NA	96.41	99.76	97.71	97.05
3	1	2.11	99.12	99.90	99.21	99.16
	8	2.11	96.82	99.70	97.22	97.02

Deploying the little network (ID 1 from table 2) with 1 250 502 weights on a Raspberry Pi 2 roughly gives an execution time of 30ms (from figure 2). This little network trained for classifying the priority class '8' has a F_1-score of 97.05% which is 0.03% better than the remote big neural network (ID 3). The overall performance of this little network is better for this specific class. Other classes yield similar results.

Forwarding the same sample input to the big network gives an execution time of 16.42ms (from table 2). From the moment the response time is more than 46.42ms (add execution time of big and little neural network) we can get a faster and equivalent response from the little neural network executed on a Raspberry Pi 2. In most cases this will not make a big difference but in highly critical situations a fast response makes all the difference.

5 Conclusion and future work

In this paper we investigated the potential of Big-Little neural network architectures, to reduce response time of the overall network, maintaining a comparable accuracy. The results show that prioritizing one class can lower the calculations required to reach the same classification performance for that class. This is only required for critical use cases where fast response time is needed. Executing a neural network is not the expensive part but the latency between the nodes is.

Important points for future work are to deduce little networks and their weights from the matching trained big network and to test the same hypothesis for convolutional neural networks on Cifar [10] and/or ImageNet [16]. In the future we will distribute neural networks on multiple IoT devices in an environment to decrease the cloud usage and increase the independence of this environment.

Acknowledgements

Part of the work was supported by the iMinds IoT research program.

References

1. Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.
2. Sean Kenneth Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 35–46. ACM, 2010.
3. Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

4. James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.
5. Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Juergen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *arXiv preprint arXiv:1003.0358*, 2010.
6. Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *Proceedings of the 30th international conference on machine learning*, pages 1337–1345, 2013.
7. Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *Industrial Informatics, IEEE Transactions on*, 10(4):2233–2243, 2014.
8. Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.
9. Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.
10. Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1(4):7, 2009.
11. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
12. Yann LeCun and Corinna Cortes. The mnist database of handwritten digits, 1998.
13. Sam Leroux, Steven Bohez, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. Resource-constrained classification using a cascade of neural network layers. In *International Joint Conference on Neural Networks*, 2015.
14. P. Parwekar. From internet of things towards cloud of things. In *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*, pages 329–333, Sept 2011.
15. Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.
16. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, pages 1–42, 2014.
17. Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61(0):85 – 117, 2015.