Deep Learning on Mobile Devices with Ubiquitous Cognitive Computing

Yueyu Hu 1400012817
huyy@pku.edu.cn

Hao Wang 1300012703
hao.wang@pku.edu.cn

ABSTRACT

The proliferation of mobile devices with increasing computing capacity and the rapid development of effective Deep Neural Networks (DNN) triggered the birth of intelligent sensing with applications like instant face recognition, indoor-outdoor detection, and personal health care. Following the design principle of human-centered philosophy, there is a need to customize one pre-trained deep model for each user. However, training or fine-tuning deep models on mobile devices is one challenging task due to the heavy resource consuming. In this paper, we proposed a distributed deep learning framework to run on mobile devices with the capacity of fine-tuning pre-trained Deep Neural Networks (DNN) models, which, to our knowledge, is the first distributed Deep Learning framework to run on mobile devices. Experiments show that our framework can successfully manage the synchronous training phase of deep models on multiple devices.

1. INTRODUCTION

Sensors are now deployed on a wide variety of devices like smartphones, tablets, cars and home appliances. Like digital cameras, built-in barometers and accelerometers and other sensors generate a huge amount of data on these devices. With the access to these statistics, analysis can be done to facilitate applications like personal health care and smart personal assistant.

With the recent development of DNN, the performance of the automatically learned feature descriptor has already outstood human hand-craft feature. The training of DNN usually requires high-end GPUs for it largely accelerates the computation of the training phase of DNNs. The computation of forwarding and backward propagation of a deep model requires intensive computation due to the large parameter space and enormous training dataset but it can be scaled up easily for largely paralleled GPU executing.

One approach to enable mobile devices to be capable of training from sensor data is to deploy computation intensive tasks onto remote servers, aka on the cloud. Raw data are sent continuously from local storage to remote server. However, there are two main problems with this kind of approach. First, the transmission consumes a large amount of bandwidth, especially for the server side. As the users accumulate, the network loads get heavier. And for the client side, which is usually a mobile device, it is usually hard to transmit raw training data of large scale in mobile networks. Second, the server should keep one unique model for each user for model optimization and customization, which could take up a large amount of storage and reduce performance. To address these problems, local deep networks are developed. The method in [4] reduce the time for computing convolutional layers on mobile devices by utilizing on-chip GPUs and doing some modification on the original complicated deep models. [7] decompose monolithic deep model network architectures into several blocks with various type and deploy those blocks to heterogeneous computing units in mobile devices. These units include CPU, GPU and even Digital Signal Processor (DSP). By performing reasonable resource scaling, this method largely improved the performance of the computing of Fully-Connected(FC) layers, which is common in DNNs. However, this kind approaches shared the problem of hardware dependency and accuracy dropping and only support the inference phase. Changes in resource availability can make big difference in the usability of these frameworks, especially in mobile devices. The ability for these models to adapt to a new environment is limited for there is little or no further training or fine-tuning process for the model on mobile devices. These static models are not suitable for applications where the models should be able to learn new knowledge, like auto-driving and behavior fitting. An example of indoor-outdoor detection [10] shows the need for on-device fine-tuning.

To address these problems, we explore a different source of computing capacity. Recently the idea of cognitive computing is gaining popularity. With low-latency local wireless network built on most of the public and private area, mobile devices are now closely connected to each other. Intuitively thinking, one task that cannot be done by one may have the chance to be done by many. Following this idea, we design the framework the Deep Learning with Ubiquitous Cognitive Computing. Here we develop a kind of distributed system on the network of personal handsets to enable on-device post-training for large-scale deep learning network. Much work has been undertaken in the past few years about deep learning on GPU clusters [2, 5] and some of the ideas have been implemented [1], which is closely related to our work. But these clusters are often well placed in a fixed environment

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

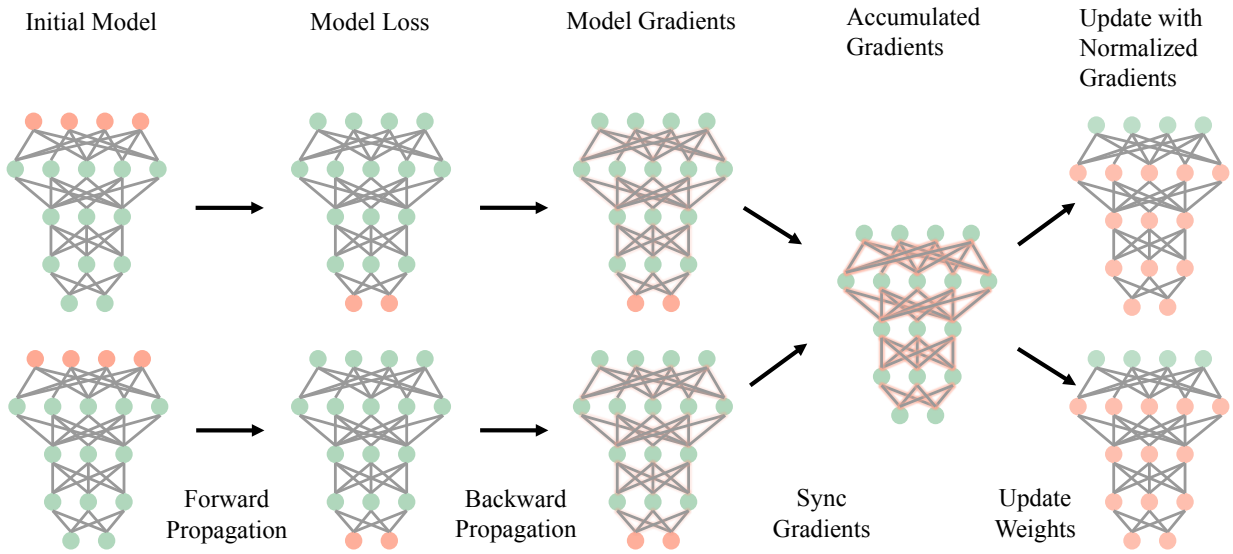

Figure 1: Framework of Training

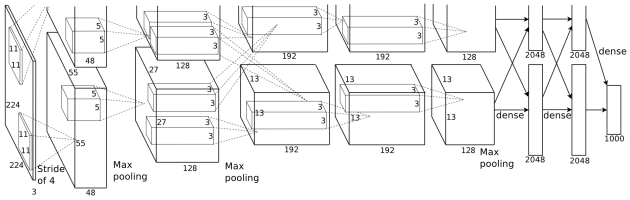

Figure 2: A typical DCNN architecture: AlexNet [6]

and some of them even share the same control apartment. For most cases, the main computing devices share data using highly efficient data link. But in handsets, the mobility determine that the network should be ad-hoc. The latency for communication would increase accordingly and this should be taken into consideration in the system design.

In the following sessions we first introduce our approach to enable parallel DNN training on mobile devices, we then introduce our synchronization design to maintain model consistency.

2. PROPOSED METHOD

In this section, we talk about the technical approach to address this problem. To enable the co-computation described earlier, it is necessary to introduce parallel computing to the design. The system is separated into two part, the parallel solver and the synchronization mechanism.

2.1 Overview of Deep Learning

Recent years, deep learning has brought a revolutionary power in amounts of the field in artificial intelligence, including visual object recognition, object detection, and speech recognition. By discovering intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters, deep learning can model high-level abstractions in data. There exist several architectures of neural networks. Deep convolutional networks have brought about breakthroughs in processing images, video, speech, and audio, whereas recurrent networks have shone the light on sequential data such as text and speech [8]. In this paper, we mainly study the deep convolutional networks.

Similar to their predecessors, shallow neural networks, deep convolutional neural networks (DCNN) consist of a large collection of neural units, which are usually arranged in several layers. DCNN are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [3] (Figure 2). The typical approach to train a DCNN model includes following steps: (1) calling network forward to compute the output and loss; (2) calling network backward to compute the gradients; (3) incorporating the gradients into parameter updates according to the solver method; (4) updating the solver state according to learning rate, history, and method. With the trained DCNN model, the common way to deploy the network is simply calling the network forward and get representations in final layer.

2.2 Solver Design

The computation of DNN solver is naturally parallel. The model is trained using the mini-batch optimization which indicates that multiple similar processes can be executed at one time. And this computation can usually be formulated as general matrix operations, which make it more scalable in a large scale parallel computing environment. In our proposed method we choose to exploit the parallelism on the solver level, which means different nodes do the forward and backward propagation with different data but update the model with the same gradient. Gradients distributed among multiple client nodes are summed and normalized on one specific node called the hub, and then the normalized gradients are scattered to client nodes for weights update.

As is illustrated in Fig. 1, we first do a forward propagation in which we generate a loss value for given batch of data.

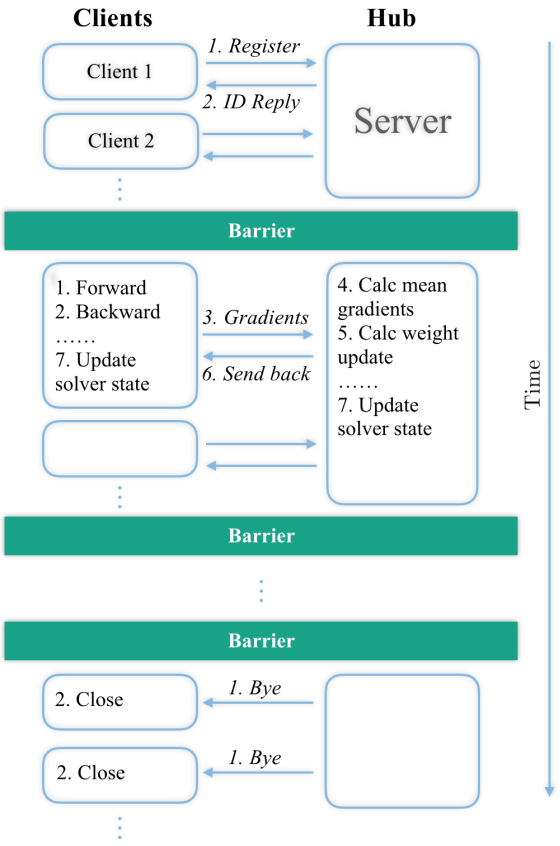

Figure 3: Synchronization Mechanism

This is done by applying the defined computational operations of each layer. When the forward propagation is done, a backward propagation is undertaken to back-propagate the loss to the very first layer for generating gradients respective to each layer. When the forward and the backward computation is done, the gradients are stored in each layer ready for the update procedure. Instead of continuing the updating process, our solver pushes the gradients to the *hub* node. A set of merged gradients of all involved nodes will be pulled from the hub to the worker node for the coming weights updating process. Finally, all the worker nodes update their weights using the provided merged gradients.

In our design, all nodes share the same model and the model is updated in a synchronous manner. All nodes keep the same model from the first time of the synchronization and keep them synchronized to the end of optimization.

2.3 Synchronization Mechanism

The other main part of the system is the synchronization mechanism, which is usually the essential part in common distributed systems. We adopt the barrier base synchronous model, in which we do synchronization in each iteration of the optimization process. The whole process is shown in Fig. 3. At the very beginning of the cooperative training, each client will send a short message to the hub for the registration. The registration process can provide enough information to select the scaling parameter during the normalization of the merged gradients. In the training process, a barrier is set before the clients' pulling of merged gradients to guarantee maintenance of the same model.

After the forward-backward process of one client, the gradients of each layer is sent from the client node to the hub node. The gradient is accumulated in the hub node and when the accumulation finishes, the normalization is exerted to the gradients. Then the gradients are broadcast to each client node.

In practice, a strategy called early-stopping is usually applied during the training process. The main idea for this strategy is to stop the training when the loss value is no longer dropping, which indicates that the model has already converged and reached the optimized state. In our synchronous model, a *goodbye* message is sent from to hub to each node to indicate the end of optimizing, and the whole training process is finally done.

Table 1: Comparation

Setting	Time for 10 Iterations. (s)
Nexus 6P only	5
Meizu MX3 only	28
Co-Training	25

3. EXPERIMENTAL DESIGN

In this section, we introduce our design and results of the experiment. We test our model using the LeNet [9] for the MNIST number figure recognition task [9]. The neural network is composed of seven layers including a convolutional layer and several fully-connected layers. This is a quite simple network compared to those sophisticated deep neural networks designed for complex task today. However, our system is designed to be compatible for all kinds of layers and networks, for the parallel part is on the solver layer of the framework.

We test our model with three smartphones running Android operating system, listed below.

- Google Nexus 6P, Qualcomm Snapdragon 820, 64-bit 2.0GHz, 4 threads
- Meizu MX 3, Samsung Exynos 5410, 32-bit 1.6GHz, 4 threads
- Google Nexus 7, NVIDIA Tegra 3, 32-bit 1.3GHz, 4 threads

Note that the Google Nexus 7 is only set as a hub node in our experiment for it is weaker in computing capacity. These devices are connected to each other within an 802.11n wireless network with an average bandwidth of 32Mbps.

The result of the experiment is shown in Table 1. From this result, we can see that, due to our synchronous model design, one drag device, which is much slower than other peers, can largely affect the performance of the whole system. However, there are several possible solutions for this. One is to modified computing workload specifically for each worker node in accordance with its hardware setting, like modifying the batch size and iteration number for one specific node.

4. CONCLUTION

In this paper, we proposed the first deep learning framework for distributed training on mobile devices. We do a solver level modification to enable sharing of gradients among multiple clients. A synchronization model featuring barriers is introduced to maintain a shared deep model. Experiments show that our model can distribute the computational workload to worker nodes in an 802.11 wireless network and support cooperative training of deep neural networks among mobile devices.

5. CONTRIBUTIONS

- Design of deep learning algorithm on mobile: Yueyu Hu, Hao Wang
- Implementation of deep learning algorithm on mobile: Yueyu Hu
- Design of synchronization algorithm: Yueyu Hu
- Implementation of synchronization algorithm: Yueyu Hu, Hao Wang
- Poster: Hao Wang, Yueyu Hu
- Report: Yueyu Hu, Hao Wang

6. REFERENCES

[1] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[2] G. Dahl, A. McAvinney, T. Newhall, et al. Parallelizing neural network training for cluster systems. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, pages 220–225. ACTA Press, 2008.

[3] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. *2015*, 2016.

[4] L. N. Huynh, R. K. Balan, and Y. Lee. Deepsense: A gpu-based deep convolutional neural network framework on commodity mobile devices. In *Proceedings of the 2016 Workshop on Wearable Systems and Applications*, pages 25–30. ACM, 2016.

[5] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W.-m. Hwu. Gpu clusters for high-performance computing. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–8. IEEE, 2009.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *NIPS*, 2012.

[7] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 1–12. IEEE, 2016.

[8] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] V. Radu, P. Katsikouli, R. Sarkar, and M. K. Marina. A semi-supervised learning approach for robust indoor-outdoor detection with smartphones. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 280–294. ACM, 2014.