Exploring the Use of Synthetic Gradients for Distributed Deep Learning across Cloud and Edge Resources

Yitao Chen
Arizona State University

Kaiqi Zhao
Arizona State University

Baoxin Li
Arizona State University

Ming Zhao
Arizona State University

Abstract

With the explosive growth of data, largely contributed by the rapidly and widely deployed smart devices on the edge, we need to rethink the training paradigm for learning on such real-world data. The conventional cloud-only approach can hardly keep up with the computational demand from these deep learning tasks; and the traditional back propagation based training method also makes it difficult to scale out the training. Fortunately, the continuous advancement in System on Chip (SoC) hardware is transforming edge devices into capable computing platforms, and can potentially be exploited to address these challenges. These observations have motivated this paper's study on the use of synthetic gradients for distributed training cross cloud and edge devices. We employ synthetic gradients into various neural network models to comprehensively evaluate its feasibility in terms of accuracy and convergence speed. We distribute the training of the various layers of a model using synthetic gradients, and evaluate its effectiveness on the edge by using resource-limited containers to emulate edge devices. The evaluation result shows that the synthetic gradient approach can achieve comparable accuracy compared to the conventional back propagation, for an eight-layer model with both fully-connected and convolutional layers. For a more complex model (VGG16), the training suffers from some accuracy degradation (up to 15%). But it achieves 11% improvement in training speed when the layers of a model are decoupled and trained on separate resource-limited containers, compared to the training of the whole model using the conventional method on the physical machine.

1 Introduction

Deep learning has spawned a large number of important applications such as augmented reality, virtual assistant, autopilot, and surveillance. Many of these applications involve the emerging smart devices on the edge, which provide training data and user interactions to deep learning. But deep learning models are typically complex and need a large training dataset to achieve good results. These requirements are only going to explode once we start learning on real-world data (e.g., camera feeds coming from thousands of places in a city). Therefore, we are facing an ever-increasing demand for computing hardware to feed and train these large, data-hungry models.

The continuous advancement of System on Chip (SoC) hardware enables edge devices to perform increasingly complex workload locally, which suggests that these devices can also be harnessed to support deep learning computations. This approach might provide the only solution to the crisis brought about by the billions of smart devices which are rapidly deployed and pervasively used in our society. Therefore, we need to find a way to train complex deep learning models on a distributed system that includes many resource-constrained edge devices.

Existing data parallelism approach is infeasible to deep learning on the edge because edge devices have only limited resources and cannot handle the full model. Instead, a more promising approach is to exploit model-level parallelism by dividing a large model into many small pieces and training them in parallel on edge devices. In this paper, we explore the use of synthetic gradients (SG) for model-parallel training of a deep neural network (DNN) model, which decouples the model layer-wise and allows the different layers to be hosted by different, resource-constrained devices. In the conventional back propagation-based training method, all the layers are tightly coupled, which forces the training to perform in a sequential synchronous manner. If one layer requires long computation time during training, then it can slow down the subsequent layers and further the overall training. In contrast, SG takes the activations from a prior layer as input and estimates the subsequent layer's error gradients, so the latter does not need to wait for the gradients from the back propagation to update the weights. Hence, layers in a model are decoupled, and each layer can be trained independently on a different device.

Initial investigation on SG has examined its performance on simple networks, providing great insights on the feasibility of SG [5, 6]. But these results are based on (1) fixed-width

networks with either only fully-connected layers or only convolutional layers and (2) Recurrent Neural Networks (RNNs), commonly used for tasks such as character level language modeling. They provide limited insights into the use of SG on real-world complex networks which use a combination of different types of layers. In comparison, this paper provides a more comprehensive study on training complex models (such as VGG16) using the SG approach. Moreover, our paper is the first to explore the use of SG to distribute training of DNNs across cloud and edge resources.

Specifically, we examine two types of network architectures for exploring the feasibility of SG-based distributed learning: (1) fixed width network models with varied depth, we evaluate a four-layer model that consists of a convolutional layer and three fully-connected layers, and an eight-layer model which consists of five convolutional layers and three fully connected layers; and (2) networks with both varied width and depth, we integrate SG into the popular, real-world model, VGG16 [12] which consists of thirteen convolutional layers (width from 64 to 256) and three fully connected layers (width 4096). To demonstrate the feasibility of deploying a model onto multiple edge devices, we also deploy our four-layer model onto four resource-limited containers using gRPC as the communication protocol among the instances.

We build the above models on TensorFlow (R1.8) [3] and evaluate them using the MNIST dataset. Our results are positive on the feasibility of using SG even for complex networks. When the number of layers increases to eight, we observe only about 2% accuracy degradation compared to the back propagation method. In the VGG16 model, the accuracy drop is more significant, about 15%. But the SG approach achieves a good speedup when each layer of a model is trained asynchronously on a container instance.

The rest of the paper is organized as follows: Section 2 introduces the background and related works; Section 3 describes the methodology of our study; Section 4 discusses the experimental results; Section 5 presents the conclusions; and Section 6 summarizes the topics for future investigations.

2 Background and Related Work

2.1 Motivations

As a motivating example, consider a smart surveillance system that can perform automated object recognition to find out anomaly behavior from the crowd. The surveillance system can adaptively optimize the global model based on ambient environments. It builds on a network of cameras, and each camera is equipped with a weak processor along with limited storage.

In a conventional cloud-only approach, all the training is performed in the cloud, which usually consists of a centralized parameter server to store the global model and a group of workers to calculate the gradients in a distributed manner. Then, each camera downloads the trained model for inference. In the smart surveillance system, there are many cameras, and each of them continuously collects high-definition videos. Training deep learning model with such data will incur heavy computation on the cloud server, while all the computing resources on the smart cameras are left wasted. In addition, when using the conventional back propagation method to train the model, the aggregated network overhead of sending gradients can also become a system bottleneck [8, 9, 11], which further limits the scalability of the training.

Instead of performing training in the centralized, cloud-only approach, we can partition the model into small portions and distribute the training tasks across the whole camera network. In this approach, each camera holds a portion of the model's layers, which is decoupled from the other layers, and perform the training of local layers independently. In this way, the training fully utilizes the available computing power on each camera and reduces the burden on the cloud server. As the training of the individual layers is decoupled, there is also no need to pass the error gradients among all the works, reducing the network overhead.

2.2 Related Work

While edge computing is still in its infancy, there are several related works on edge-based distributed deep learning. Federated learning [7, 13] provides a solution for training a large, global model on a centralized server by federating many small, distributed models on edge devices. While both exploiting the capabilities of edge devices in training, federated learning and our synthetic gradient approach have different objectives. Federated learning focuses on preserving user privacy in training since all the user data is used only locally for training the local, small model. In comparison, our SG approach focuses on achieving model-parallelism by training different layers of a model asynchronously. Federated learning still needs heavy computation in the cloud to calculate the aggregated gradient when training the large, global model. The SG approach can also help federated learning by further breaking down the training and performing it in parallel on different devices.

There are several related works on alternative techniques to the conventional back propagation in training DNNs. For example, feedback alignments (FA) [10] use a random weight matrix instead of the transposed weight matrix from the forward path to pass the error signal and help improve the model. Kickback (KB) [4] collects the error gradient only from targeted adjacent neurons in the upstream layer to speed up the lengthy back propagation. However, these works still rely on synchronous update on the weights of the downstream layer. In comparison, the SG approach uses another simple neural network to predict the error gradient during the back propagation, and thus each layer is no longer linked together by the error gradients, allowing the decoupling of layers in a model.

As discussed in the introduction, there are some preliminary

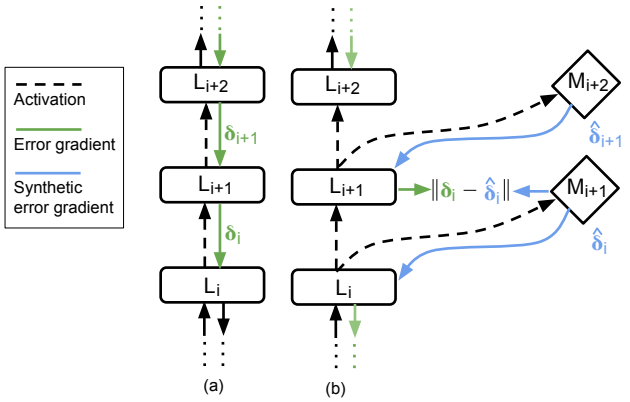


Figure 1: (a) Data flow of training a model with conventional back propagation. The need of error gradients (δ_i and δ_{i+1}) to proceed leads to the backward coupling. The L_{i+1} needs to update its weights and then calculate for the δ_i. (b) Data flow of training a model with SG. The backward coupling is removed because the model can rely on the error gradients ($\widehat{\delta_{i+1}}$ and $\widehat{\delta_i}$) provided by the SG modules (M_{i+2} and M_{i+1}) to proceed.

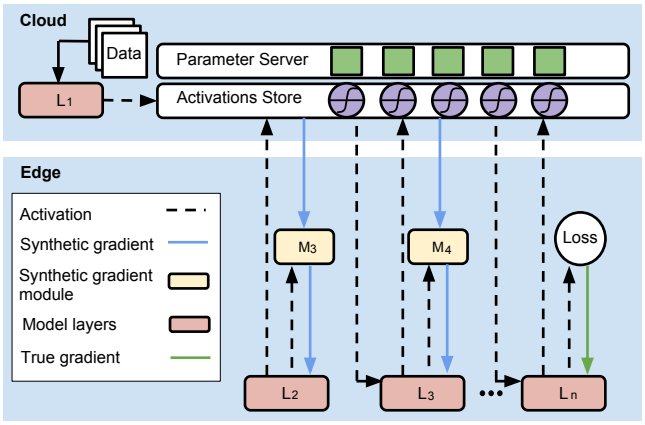


Figure 2: Cross-edge-cloud collaborative training of a deep neural network. Each layer of a large model can be hosted on potentially less powerful devices.

works on the use of SG to achieve model level parallelism, but their scope and results are limited. Our study provides a more comprehensive investigation on this approach by considering complex networks with different types of layers. Moreover, we also explore the use of edge devices in distributed training, which is an important and unexplored application of the SG approach. The rest of this paper describes the details of our methodology and results.

3 Design

In this paper, we envision a novel distributed training paradigm which leverages model level parallelism to train a complex model on the edge devices and the cloud collaboratively. Each layer of a complex model is held and trained independently on a less powerful edge device.

3.1 Synthetic Gradients

Figure 1 shows the data flow of using SG modules to decouple the layers in a model as compared to the back propagation method. We take a look at a three-layer subgraph of a model, L_i, L_{i+1}, and L_{i+2}, connecting in sequential order. The output of layer L_i is h_i, i.e., the activation. h_i is passed to L_{i+1} as well as the SG module M_{i+1}, which outputs the prediction of the true error gradient from layer L_{i+1}, $\widehat{\delta_i}$, to update L_i. The SG module is another neural network with one hidden layer and needs training to make a good prediction of the true error gradient. To train the SG module M_{i+1}, we need to provide the true error gradient δ_i. Instead of waiting for the true error gradient to back propagate, this approach uses the synthetic error gradients from M_{i+2} to train M_{i+1}. The synthetic error gradient $\widehat{\delta_{i+1}}$ generated by M_{i+2} back-propagates to layer L_{i+1} to generate δ_i which is then used to train M_{i+1}. As we can see, the "true gradient" may be different from the gradient coming from the back propagation. Hence, using the error gradient predicted by another SG module as the true gradient is a crucial approximation to completely remove the backward coupling. But the approximation also introduces some potential training quality degradations to the neural network.

3.2 SG-based Cross-cloud-edge Distributed Training

Edge devices are usually resource constrained compared to cloud servers, so we assume that the edge devices satisfy the following three constraints: (1) each device cannot hold the whole model in its storage; (2) each device has enough storage to hold at least two layers of the model; and (3) each device may have different hardware capabilities.

Figure 2 illustrates the architecture of our cross-cloud-edge distributed training scheme, which consists of four major logical components. (1) Parameter server stores all the weights for a global model. (2) Activation layer stores the activations from different layers and sends them to the SG modules. (3) Each SG module takes the activations from the previous layer to predict the gradient input for the training to proceed without waiting for the previous layer to complete. (4) Each device holds two network model layers, the layer from the full model and the layer from the SG module. Note that a device can hold more than two layers and some of the model layers can also be held and trained in the cloud, depending on the actual resource availability. The architecture illustrated in Figure 2 is simplified for the sake of a concise discussion.

Our model-level parallelism approach takes advantage of

SG to eliminate the layer coupling in training a deep neural network model using the conventional back propagation method [5,6]. As an example shown in Figure 2, the SG module M_3 receives the activation from L2 and then generates the SG $\widehat{\delta_2}$. M3 also collects the gradient δ_2 from the cloud to train the gradient prediction.

The significance of model-level parallelism in distributed training is two fold. First, computation efficiency: we can utilize the idle devices to work with the cloud on training computationally expensive models. Second, communication efficiency: since we do not need to continuously transfer the gradients to the cloud for updating the model, we can substantially reduce the network overhead and free up the valuable bandwidth for other uses. Both computation and communication efficiency are crucial to the scalability of training complex models using massive data (contributed by the exponentially growing number of smart devices deployed on the edge).

4 Evaluation

We present our preliminary results to answer the following research questions: (1) Is there any performance degradation from using the SG approach as compared to the conventional back propagation method? We discuss the results to this question in Sections 4.1 and 4.2; (2) Compared to the conventional back propagation method, is there any improvement in training speed if the model is trained asynchronously with the SG approach? Section 4.3 discusses the training speed results.

We use three different models implemented with TensorFlow (R1.8), (1) a four-layered model, which consists of one conventional layer and three fully-connected layers. The conventional layer is followed by a max pooling layer to reduce the dimension of the input; (2) an eight-layered model, which consists of five conventional layer and three fully-connected layers; and (3) the VGG16 [12] model, which has 13 conventional layers and 3 fully-connected layers. The width of conventional layers (the number of channels) starts from 64 and increases by a factor of two after each pooling layer until 512. The SG modules are added to the output of each pooling layer. The width of the SG module follows the increment fashion as the conventional layers in the model, starting from 64 and ending with 512. For all the cases, the SG module is conditioned on the labels, which concatenates the labels and the input of the SG to increase the performance of the training [5].

We conduct all the experiments on a server equipped with Nvidia K40 GPU and Xeon CPU, and we use multiple Docker containers configured with limited CPU and memory to emulate the deployment of our approach on resource-constrained devices. The limitation of our experiment setup is that the neural network models are trained on a single machine, which cannot practically capture the network communication when training asynchronously among different edge devices. Since the communication should not impact the accuracy and the convergence speed, we believe that the results can still provide good insights into designing model-parallel distributed training using the SG approach. This preliminary study is our first step towards distribute deep learning across cloud and edge resources.

	BP	**SG**
Four-layered model	0.984	0.977
Eight-layered model	0.992	0.924
VGG16	0.994	0.845

Table 1: Accuracy values comparison between training with conventional back propagation and synthetic gradient.

4.1 Accuracy

We measure the accuracy of the three models from using the SG approach and compare it with the conventional back propagation (BP) method. Table 1 lists the accuracy results for the above models.

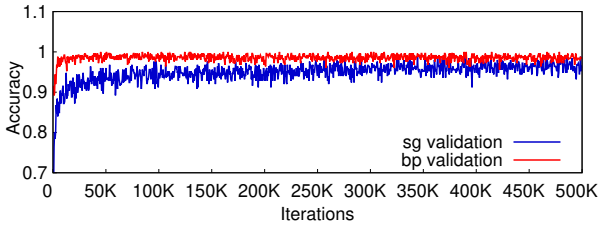
The result shows that SG can reach comparable accuracy to BP when the model is shallow and not complex. In the four-layer case, SG has less than 1% accuracy loss compared to the back propagation method. In the eight-layer case, the accuracy gap between SG and BP becomes larger. When the model depth increases to eight, BP reaches 99.2% accuracy but SG stops at 92.4%, after 500K iterations of training. The eight-layered model has an accuracy gap with BP that is 5% wider than that of the four-layered model. The result shows that model depth has an impact on accuracy when using the SG approach. We believe the reason is that the error of the prediction from each synthetic module accumulates and hence affects the overall accuracy.

The accuracy degradation becomes even more significant when the model becomes more complex and has convolutional layers with different widths. The accuracy gap between SG and BP in the VGG16 model is about 15%. The complexity of the network model reduces the accuracy of the SG method. As the SG module may lose useful features when predicting the error gradients, we need to further optimize the hypermeters in the SG modules to address the accuracy drop.

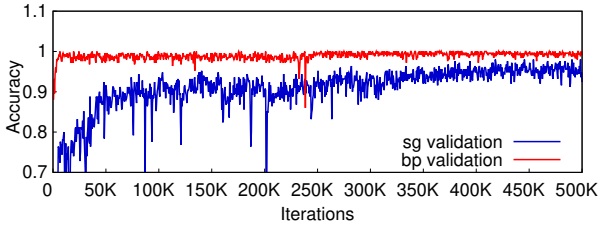
4.2 Convergence Speed

We next look into the impact on the convergence speed when increasing the model depth. Convergence speed measures how fast a model can reach a satisfactory level of accuracy during training. We compare the convergence speed using SG vs. conventional back propagation.

Figure 3a shows that the SG approach is learning slower than the conventional back propagation method. We notice that the slope of the SG result is less steep compared to BP,



(a) Four-layer model accuracy comparison



(b) Eight-layer model accuracy comparison

Figure 3: Accuracy comparison between the synthetic gradient (SG) and the conventional back propagation (BP) method on a four-layer model and a eight-layer model. We only show the training result on the 500K iterations configuration.

which means that rate of converting learning into accuracy is slower. The SG accuracy reaches more than 90% accuracy after 50K iterations. After that the slope of the accuracy is gradually stabilized.

Figure 3b shows that the increase of the number of layers in a model can further slow down the converge speed of the SG approach. The most significant improvement of accuracy happens within the first 100K iterations, which reduces the accuracy gap between SG and BP to about 10%. The eight-layer model converges about 2X slower than the four-layer model. The SG approach creates a trade-off between the convergence speed and layer-independence. Intuitively, the more SG modules that we introduce into the system, the longer time it will take for the model to converge.

4.3 Emulated Deployment

We deploy our four-layered model layer-wise on four Docker container [1] instances to emulate distributed training on edge devices. Each instance is configured with limited resources (one CPU and 1GB memory) and is responsible for training one layer of the model. The communication between workers uses gRPC [2]. We measure the training speed in terms of iterations per second. During each iteration, the model processes one batch of images in the training dataset. Our batch size is 256 and the learning rate is initialized as 3×10^{-5} as suggested in the related work [6].

We observe that the training speed has a slight improvement, about 11%, when training in a layer-wise decoupled fashion on four container instances. The training speed is on average 185 iterations per second when training layer-wise, as compare to 166 iterations per second when training using the conventional back propagation method on the same hardware platform but without using containers.

5 Conclusions

In this paper, we explore the use of synthetic gradients for distributed deep learning across cloud and edge. We implement asynchronous training using SG to decouple the layers of a model, deploy them on different devices, and achieve model level parallelism. We demonstrate the feasibility of this approach and evaluate its accuracy and convergence speed on various model configurations. Our results show comparable performance with the conventional back propagation method. When the model is more complex, we do observe some accuracy degradation as compared to the conventional back propagation method. In addition, we demonstrate the deployment of this approach by using container instances to emulate edge devices, which also shows positive results with a 11% improvement on training speed for a four-layer model.

6 Discussion Topics

This paper has shown promising preliminary results and revealed the potential of using synthetic gradients to distribute deep learning across edge and cloud. Many exciting research questions can be explored in our future work, including:

- How to further reduce the accuracy degradation in using our SG approach to train complex deep learning models? We observe more accuracy degradation in the VGG16 result compared to the eight-layer model result. It is important to reduce the accuracy degradation so that SG approach can be applied to more complex models.

- How to improve the convergence speed for the SG approach to reach good accuracy quickly? Figure 3 illustrates the slowdown of an eight-layered model compared to a four-layered model; we believe it is crucial to reduce the convergence time so that the model can learn faster.

- How to design efficient compression techniques to further reduce the communication frequency between the SG module and activation store? In our SG approach, each SG module receives activation from the activation store during the training process. The communication between the SG module and the cloud incurs communication overhead. We can further reduce the communication overhead using compression techniques to reduce the size of each activation, or we can carefully control how frequently communication happens.

- How to appropriately partition a model based on the capability of different edge devices? Edge devices have

diverse hardware capabilities. To reduce the overall training time, we need to consider how many layers each device should hold.

- How to design an activation store and parameter server that can provide efficient access to the SG modules? The activation store contains the layer output from part of a model and sends the activation to the synthetic gradient as input. It is critical for the activation store to support fast access by the SG modules.
- Evaluate the effectiveness of using our approach to train more complex networks with large datasets.
- Evaluate the deployment of our approach on actual edge devices and consider the actual communication links between the devices and the cloud.

7 Acknowledgment

This research is sponsored by U.S. National Science Foundation CAREER award CNS-1619653 and awards CNS-1562837, CNS-1629888, IIS-1633381, and CMMI-1610282. We thank our shepherd, Irfan Ahmad, and the anonymous reviewers for their helpful insights.

References

[1] Docker. https://www.docker.com/, 2013.

[2] Google gRPC. https://grpc.io/, 2015.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[4] David Balduzzi, Hastagiri Vanchinathan, and Joachim Buhmann. Kickback cuts backprop's red-tape: biologically plausible credit assignment in neural networks. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[5] Wojciech Marian Czarnecki, Grzegorz Swirszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 904–912. JMLR.org, 2017.

[6] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1627–1635. JMLR. org, 2017.

[7] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[8] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, 2014.

[9] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[10] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276, 2016.

[11] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017.