San Jose State University

CS218 Project Outline

Large Scale Distributed Deep Networks

Siddharth Kumar
siddharth.kumar@sjsu.edu

Krishna Chaitanya Mullapudi
krishnachaitanya.mullapudi@sjsu.edu

October 22, 2017

Contents

1 Introduction

In machine learning, accuracy of a model increases with the increase in the training examples and model parameters [4],[6]. These have led to a drastic scaling up of the training and inference algorithms used for these models [8]. The use of GPUs made it more practical to train deep neural networks. Although even for GPUs the training speed-up is small when the model cannot fit in the GPU memory. Therefore to use GPU effectively, researchers usually reduce the size of the data or parameters so that CPU to GPU data transfer is not a bottleneck. While such an approach may work for small problems, such as acoustic modeling for speech recognition, they render ineffective for problems with large number of training examples and dimensions.

In this project we investigate an alternative approach, using large clusters of machines to distribute training and testing in deep networks. We begin by studying the software framework called *DistBelif*. It is frmework for distributed modeling of deep neural networks, develop by the Google Brain project [1]. It not only enables parallelism for deep learning models within a machine using multithreading but also and across multiple machines using message passing and synchronization. We will emphasize on the details of the parallelism, synchronization and communication implemented by the framework. Then we will examine Downpour SGD, an asynchronous stochastic decent procedure that uses adaptive learning rates and supports a large number of model replicas. With many improvements over DistBelif, Google came up with the second generation of their machine learning framework TensorFlow. We examine the distributed implementation of the TensorFlow framework. TensorFlow takes the computations described as data-flow like model and maps them into a wide variety of different hardware platforms. This allows us to inference our models on mobile platforms such as Android or iOS, individual machines containing single or multiple CPU cards to large-scale distributed systems running on specialized machine or the cloud. We investigate in detail the programming model of the TensorFlow interface and both the single and distributed implementations. Then we use the distributed configuration of TensorFlow to to train a convolutional neural network using the MNIST dataset on the Google could machine learning engine. Use TensorBoard to visualize the training process and then analyze the models in terms of classification and detection accuracy.

[1] https://en.wikipedia.org/wiki/Google$_Brain$

2 DistBelif

DistBelief was used by Google and its subsidiaries for models of images classification[5], object detection [9], speech recognition [10], pedestrian detection [1], reinforcement learning [7] and many other fields. Most of the commonly used applications like Google search, Google Maps, Google Photos, Google Translate, YouTube and other, used the DistBelief framework to deploy deep neural networks.

2.1 Architecture

The user defines the computation that takes place at each node and each layer of the model and the messages that should be passed in the upward and the downward phase of the computation. The improvement in performance of a distributed deep network depends on the connectivity structure and the computational need of the model. Typically, when the models have high computational demand or large number of parameters, the communication overhead between the nodes becomes insignificant and training and inference is much faster due to access to more CPU and memory. DistBelief not only parallelizes the computation within a single model, but it distributes training across multiple model instances. We consider the Downpour SGD a DistBelief implementation of Stochastic gradient decent (SGD) to understand the architecture of DistBelief.

2.2 Downpour SGD

Stochastic gradient decent (SGD) is one of the most commonly used optimization procedures for training deep neural nets [2],[3]. Downpour SGD is a variant of asynchronous SGD that uses multiple replicas of the same DistBelief model. The training data is divided into a number of subsets and a copy of the model is run on each of these subsets. A centralized parameter server stores the current state of all the model parameters, shared across many machines. Before processing each bach of data, the model replica asks the parameter server for the latest copy of its model parameters. Each model communicates with the subset of the parameter server that holds the parameters relevant to its partition.

3 TensorFlow

TensorFlow is a second-generation system for the implementation and deployment of large scale machine learning models. Compared to DistBelief, TensorFlows programming model is more flexible, its performance is significantly better, and it supports training and using a broader range of models on a wider variety of heterogeneous hardware platforms. TensorFlow takes computations described using a dataflow-like model and maps them onto a wide variety of different hardware platforms, ranging from running inference on mobile device platforms such as Android and iOS to modestsized training and inference systems using single machines containing one or many GPU cards to large-scale training systems running on hundreds of specialized machines with thousands of GPUs. TensorFlow allows clients to easily express various kinds of parallelism through replication and parallel execution of a core model dataflow graph, with many different computational devices all collaborating to update a set of shared parameters or other state.

3.1 Basic Concepts of Tensor Flow

Graph in a Tensor Flow is a representation of a dataflow computation, composed of a set of *nodes*. Clients typically construct a computational graph using one of the supported frontend languages (C++ or Python).

3.2 Architecture

The main components in a TensorFlow system are the client, which uses the Session interface to communicate with the master, and one or more worker processes, with each worker process responsible for arbitrating access to one or more computational devices (such as CPU cores or GPU cards) and for executing graph nodes on those devices as instructed by the master. We have both local and distributed implementations of the TensorFlow interface. The local implementation is used when the client, the master, and the worker all run on a single machine in the context of a single operating system process (possibly with multiple devices, if for example, the machine has many GPU cards installed). The distributed implementation shares most of the code with the local implementation, but extends it with support for an environment where the client, the master, and the workers can all be in different processes on different machines. In our distributed environment, these different tasks are containers in jobs managed by a cluster scheduling system.

4 Neural Networks

A neuron is the basic unit of a Neural Network. A neuron performs a dot product with the input values and its weights, adds the bias and applies the activation function, in our case we use the sigmoid function $\sigma(x) = 1/(1 + \exp^{-x})$, as the activation function as it is the most commonly used. Neural Networks are modeled as collection of neurons that are connected in an acyclic graph. Cycles are not allows as this would imply infinite loop in the forward pass of a network. The neurons are often organized into distinct layers, the most layer type for regular neural network is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neuron within a single layer has no connections.

5 Convolution Neural Networks (CNNs)

Convolutional Neural Networks are very similar to ordinary Neural Networks, they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function on the last (fully-connected) layer.

So what does change? CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

6 CNN on the Cloud

Here we will use the distributed configuration of TensorFlow code in Python on Google Cloud Machine Learning Engine to train a convolutional neural network (CNN) model using the MNIST dataset. Then we will use the TensorBoard to visualize the training processes and compare it with other distributed CNNs in terms of training times, inference times and classification accuracy.

6.1 Google Cloud Machine Learning Engine

The Google Cloud Machine Learning Engine[2] combines the infrastructure of the Google Cloud Platform with flexibility of TensorFlow for distributed modeling of deep neural nets (DNNs). It can used to model and scale DNNs and inference from the new data coming to the cloud. In other words, it allows end-to-end deployment of machine learning models in the cloud.

6.2 MNIST Dataset

The MNIST database of handwritten digits[3], available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. We have chosen MNIST dataset because it has very low testing error rate for CNNs and requires minimal effort for preprocessing.

7 Conclusion

In this project we study the architecture of DistBelief a framework for distributed modeling and inference on deep neural networks. Neural Networks are systems that can be used for classification of data like image into classes, but are manageable only for small images, for larger images we use CNNs, where in each layer the neurons are modeled in three dimensions. This design is specially effective for images. After this, we also explore how TensorFlow, improves DistBelief and allows us to train models in a faster way on heterogeneous distributed systems and the cloud. We will use TensorFlow to train a CNN using the MNIST dataset on the Google cloud platform.

[2] https://cloud.google.com/ml-engine/
[3] http://yann.lecun.com/exdb/mnist/

References

[1] Anelia Angelova, Alex Krizhevsky, and Vincent Vanhoucke. Pedestrian detection with a large-field-of-view deep network. In *Proceedings of ICRA 2015*, 2015.

[2] Lon Bottou. Stochastic gradient learning in neural networks. In *In Proceedings of Neuro-Nmes. EC2*, 1991.

[3] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *CoRR*, abs/1003.0358, 2010. URL: `http://arxiv.org/abs/1003.0358`.

[4] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL: `http://proceedings.mlr.press/v15/coates11a.html`.

[5] Andrea Frome, Greg Corrado, Jonathon Shlens, Samy Bengio, Jeffrey Dean, MarcAurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. In *Neural Information Processing Systems (NIPS)*, 2013.

[6] Quoc V. Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y. Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 265–272, 2011.

[7] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015. URL: `http://arxiv.org/abs/1507.04296`.

[8] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 873–880, New York, NY, USA, 2009. ACM. URL: `http://doi.acm.org/10.1145/1553374.1553486`, `doi:10.1145/1553374.1553486`.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL: `http://arxiv.org/abs/1409.4842`.

[10] Matthew D. Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Mark Z. Mao, K. Yang, Quoc Viet Le, Patrick Nguyen, Andrew W. Senior, Vincent Vanhoucke, Jeffrey Dean, and Geoffrey E. Hinton. On rectified linear units for speech processing. In *ICASSP*, pages 3517–3521. IEEE, 2013. URL: `http://dblp.uni-trier.de/db/conf/icassp/icassp2013.html#ZeilerRMMYLNSVDH13`.