Sample paper

The following is from:

http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf

Introduction

Deep learning and unsupervised feature learning have shown great promise in many practical applications. State-of-the-art performance has been reported in several domains, ranging from speech recognition [1, 2], visual object recognition [3, 4], to text processing [5, 6]. It has also been observed that increasing the scale of deep learning, with respect to the number of training examples, the number of model parameters, or both, can drastically improve ultimate classification accuracy [3, 4, 7]. These results have led to a surge of interest in scaling up the training and inference algorithms used for these models [8] and in improving applicable optimization procedures [7, 9]. The use of GPUs [1, 2, 3, 8] is a significant advance in recent years that makes the training of modestly sized deep networks practical. A known limitation of the GPU approach is that the training speed-up is small when the model does not fit in GPU memory (typically less than 6 gigabytes). To use a GPU effectively, researchers often reduce the size of the data or parameters so that CPU-to-GPU transfers are not a significant bottleneck. While data and parameter reduction work well for small problems (e.g. acoustic modeling for speech recognition), they are less attractive for problems with a large number of examples and dimensions (e.g., high-resolution images). In this paper, we describe an alternative approach: using large-scale clusters of machines to distribute training and inference in deep networks. We have developed a software framework called DistBelief that enables model parallelism within a machine (via multithreading) and across machines (via 1 message passing), with the details of parallelism, synchronization and communication managed by the framework. In addition to supporting model parallelism, the DistBelief framework also supports data parallelism, where multiple replicas of a model are used to optimize a single objective. Within this framework, we have designed and implemented two novel methods for large-scale distributed training: (i) Downpour SGD, an asynchronous stochastic gradient descent procedure which leverages adaptive learning rates and supports a large number of model replicas, and (ii) Sandblaster L-BFGS, a distributed implementation of L-BFGS that uses both data and model parallelism.1 Both Downpour SGD and Sandblaster L-BFGS enjoy significant speed gains compared to more conventional implementations of SGD and L-BFGS. Our experiments reveal several surprising results about large-scale nonconvex optimization. Firstly, asynchronous SGD, rarely applied to nonconvex problems, works very well for training deep networks, particularly when combined with Adagrad [10] adaptive learning rates. Secondly, we show that given sufficient resources, L-BFGS is competitive with or faster than many variants of SGD. With regard to specific applications in deep learning, we report two main findings: that our distributed optimization approach can both greatly accelerate the training of modestly sized models, and that it can also train models that are larger than could be contemplated otherwise. To illustrate the first point, we show that we can use a cluster of machines to train a modestly sized speech model to the same classification accuracy in less than 1/10th the time required on a GPU. To illustrate the second point, we trained a large neural network of more than 1 billion parameters and used this network to drastically improve on state-of-the-art performance on the ImageNet dataset, one of the largest datasets in computer vision.

In recent years commercial and academic machine learning data sets have grown at an unprecedented pace. In response, a great many authors have explored scaling up machine learning algorithms through parallelization and distribution [11, 12, 13, 14, 15, 16, 17]. Much of this research has focused on linear, convex models, where distributed gradient computation is the natural first step. Within this area, some groups have relaxed synchronization requirements, exploring delayed gradient updates for convex problems [12, 17]. In parallel, other groups working on problems with sparse gradients (problems where only a tiny fraction of the coordinates of the gradient vector are non-zero for any given training example) have explored lock-less asynchronous stochastic gradient descent on shared-memory architectures (i.e. single machines) [5, 18]. We are interested in an approach that captures the best of both worlds, allowing the use of a cluster of machines asynchronously computing gradients, but without requiring that the problem be either convex or sparse. In the context of deep learning, most work has focused on training relatively small models on a single machine (e.g., Theano [19]). Suggestions for scaling up deep learning include the use of a farm of GPUs to train a collection of many small models and subsequently averaging their predictions [20], or modifying standard deep networks to make them inherently more parallelizable [21]. Our focus is scaling deep learning techniques in the direction of training very large models, those with a few billion parameters, but without introducing restrictions on the form of the model. In special cases where one layer dominates computation, some authors have considered distributing computation in that one layer and replicating computation in the remaining layers [5]. But in the general case where many layers of the model are computationally intensive, full model parallelism in a spirit similar to [22] is required. To be successful, however, we believe that model parallelism must be combined with clever distributed optimization techniques that leverage data parallelism. We considered a number of existing large-scale computational tools for application to our problem, MapReduce [23] and GraphLab [24] being notable examples. We concluded that MapReduce, designed for parallel data processing, was ill-suited for the iterative computations inherent in deep network training; whereas GraphLab, designed for general (unstructured) graph computations, would not exploit computing efficiencies available in the structured graphs typically found in deep networks.