Abstract

Introduction

Related work

Parallelism

To speed up neural network training, many parallelism methods have been used widely. Data parallel is the straightforward method to deal with large amount of data. With the algorithm of stochastic gradient decent, the training set would be split into small batch, and train the model batch by batch. The entire model sends to all training nodes, using different data batches run forward and backward propagation, and finally synchronous the weight, or gradient between all nodes in the cluster to update the model. There are many methods to deal with the synchronization issue.

Synchronous asynchronous synchronous with back up worker

Model parallelism is different form data parallelism. This method is aimed to split model into several small parts and distribute them on different nodes. The data flow will go through all the nodes, determined by the graph. Every node would only compute part of the entire model and no need to synchronize weights and gradients. However, compared to data parallelism, load balancing is much more difficult to maintain with model parallelism. the other issue is the transmission data size. In model parallelism we transmit feature maps between nodes instead of weights or gradients. The size of feature maps usually much larger than the size of weight or gradient, so the communication environment may be considered as a bottleneck of training speed.

Low transmission speed, low computation power, and low memory usage are the properties in modern edge devices. To speed up inference time on edge devices, there are many solutions on both hardware and software. On the hardware side, AI chips are designed for speeding up specific operations used by neural networks. On the software side, pruning and quantization are most usage method to reduce computations. A compiler or an IR can also optimize the computation graph. However, parallelism, the most straightforward method is seldom to be considered in inference scenarios caused by some difficulties. In many scenario of inference on edge, sensors receive some kind of data, send to backend to run a specific task by a pre-trained model after preprocessing. Most of the data might be past one by one, not batch by batch, so data parallelism cannot be used. Some models are well designed with parallel path in their computation graph. However, it would encounter another problem – synchronization. When training, there are lots of solutions to deal with synchronization, such as parameter server, backup worker, stale synchronization, etc. These methods are hard to be deployed on edge due to the large overhead on transmission.

Method

As mention above, low-cost is a main property of edge devices, and it comes with many constraint of resources. The main idea of distributed inference is balancing the time spent between transmission and computation. Consider a two path computation graph

Experiments

-dataset

-data augmentation

Reference