CIS631 Parallel Processing: Project Proposal

Isaac Ahern, Trevor Bergstrom, Adam Noack

October 9th, 2019

1 Introduction / background

In recent years, deep neural networks (DNNs) have been shown to be useful for solving complex problems across a wide array of domains. These networks are typically utilized when dealing with supervised learning problems, and involve learning a parametrized model $f : \mathcal{X} \to \mathcal{Y}$. The learning process for such models involves intensive computation, as an intermediary form of the model is used for inference on the data, and model parameters θ are adjusted iteratively during backpropogation based on the model loss function $\mathcal{L}(f(\mathbf{X}), \mathbf{Y}, \theta)$. In order to make training feasible over complex model architectures and large datasets, the computations $f(\mathbf{X})$ and $\partial_\theta f(\mathbf{X})$, for the forward and backward pass respectively, must be made tractable.

2 Problem formulation / motivation

It is often the case that models need to train over many iterations with large quantities of data to achieve accurate results. During this training phase a large amount of computational power is used, and as a consequence, training times can take days or even weeks to complete. Because of this, practitioners are at times forced to settle for less than optimal model architectures because training larger or more complex models would be impractical. Nevertheless, many of the operations used to train and evaluate DNNs are parallelizable. Thus, with proper programming techniques, bigger and better DNNs can be built.

There are two parallelism schemes that are frequently used to speed up training of DNN models, data parallelism and model parallelism [1].

2.1 Data Parallelism

With the ever increasing amounts of labeled data available for training DNN models, computation time and complexity increases. One way to handle the large amounts of training data is to utilize multiple hardware resources. It is not uncommon to train large models on multiple GPUs.

Using a single GPU, it is common that the whole data set cannot fit into memory, thus the data will need to be broken up in batches for processing.

The downfall of this is that when preforming stochastic gradient descent on this smaller batch of data, it is possible that the estimate of the gradients does not represent the gradient of the full data set.

Utilizing parallelization in the processing of the training data, the dataset is split into multiple small batches and sent off to multiple nodes to compute the gradients. These gradients are reported back to the main computation node. Then the weighted average of these gradients is used to update the model during backpropagation, based on the decomposition over batches

$$\overline{\partial_\theta f^{\mathbf{X}}} := \underbrace{\frac{1}{n}\sum_{x\in\mathbf{X}}\frac{\partial f(x)}{\partial\theta}}_{\text{mean "}\partial_\theta f\text{" on }\mathbf{X}} = \sum_{i=1}^{|P_{\mathbf{X}}|} \underbrace{\frac{|\mathbf{B}_i|}{n}}_{\text{weights }\lambda_i} \left[\underbrace{\frac{1}{|\mathbf{B}_i|}\sum_{x\in\mathbf{B}_i}\frac{\partial f(x)}{\partial\theta}}_{\text{mean "}\partial_\theta f\text{" on batch }\mathbf{B}_i}\right] = \sum_{i=1}^{|P_{\mathbf{X}}|}\lambda_i\overline{\partial_\theta f^{\mathbf{B}_i}}.$$

($\mathbf{X} = \bigsqcup_i \mathbf{B}_i$ is a partition $P_{\mathbf{X}}$ of $\mathbf{X}$ into batches $\mathbf{B}_i$)

2.2 Model Parallelism

DNNs are commonly being used to solve more complex problems, and these problems require large networks with many hidden layers. Growing the size of a network requires a larger memory footprint that can quickly surpass the memory on a single node, especially when you take into account the massive amount of data that needs to share the same memory.

Model parallelism is a method of dividing up a large model between multiple nodes. This requires splitting the the model up into batches of consecutive layers where each node performs the forward pass in sequence. At the end of the forward passes, the backpropagation is computed in reverse order across the nodes.

3 Directions of investigation / expected results

For this project, we will investigate how data and model parallelism schemes increase training performance of a DNN used to classify handwritten digits using the standard MNIST dataset. We will measure processing speed as each model is trained and also record how long it takes for each model's parameters to converge to the optimal values. For data parallelism tests, we can compare training time and performance of models with parallelized and non-paralellized gradient computation. Specifically, we can compare the speedup over models trained with nonparallelized gradients, but where the entire gradient is computed in series, and we can compare the performance in terms of number of epochs required for the loss to converge or hit a certain threshold for models trained with nonparallelized gradients that perform stochastic gradient descent using the largest data subset which fits in memory.

References

[1] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *CoRR*, abs/1802.09941, 2018.