CMSC416: Introduction to Parallel Computing

Topic: Deep Learning in Parallel
Date: May 02, 2024

Deep Neural Networks:
Deep Neural Networks are characterized by multiple layers of neurons. Each neuron takes an input and performs a mathematical operation on it and produces an output. DNNs are the core of a variety of machine learning related tasks, such as object detection in images and videos, and language learning models (e.g. Chat GPT).
Let's take binary classification of an object for example. The task is to figure out if this image is a cat or a dog. There are a variety of steps to take in order for our model to provide an accurate output.

1. Train the model: We do this by calculating weights for our model.
    a. Forward Propagation: Pass weights through the model and do calculations through each layer of the network
    b. Backward Propagation: Based on the loss (a function that calculates the difference between the model's guess and the actual answer) we adjust weights and pass those weights back through the model.
        i. Gradient Descent: Update weights based on the derivative of the loss function and the learning rate
    c. This is done over and over again a set number of times. The goal is to minimize loss. Minimizing loss leads to a higher accuracy. We can organize the processing of the data.
        i. Batches: A subset of the dataset that will be iteratively processed
        ii. Epochs: One epoch is one pass over all batches.
    d. Parameter tuning: We can modify the batch size, number of epochs, and learning rate in order to get as good of an accuracy as we can with the model we are using.
2. Test the model: Compare the outputs of our trained model with factually correct outputs.

These methods are not only used for binary classification, but are a general guideline on training neural networks.

Parallel/Distributed Training:
Deep Neural Networks can contain millions and billions of parameters and it would take years or even lifetimes to train if not done in parallel. By using parallel methods, we can cut training time drastically.
For example, we can train models on an Nvidia GPU by using CUDA. We don't need to limit ourselves to just one GPU. Thousands of GPUs are used to train models with many parameters. Let's take a look at some ways that we can train our model.

- **Data Parallelism:**
  Distribute data among multiple workers. Workers can be GPUs or CPUs. Each worker will have its own copy of the neural network.

    - Each worker will do a forward pass, loss calculation, and backward pass through its copy of the neural network.
    - Then do an all reduce to synchronize the gradients
    - Pros are that it is embarrassingly parallel and that it is easy to use
    - However you cannot train models that exceed the memory capacity of a single GPU.
- **Inter-layer Parallelism:**
  Distribute layers among multiple workers.
    - We can have concurrent computation of layers among multiple processes in combination with parallel processing within neurons
    - Pipelining: Each process does computations and the output of that layer is input into the next layer in a pipeline fashion
- **Intra-layer Parallelism:**
  Divide the work of an individual layer among multiple workers.
    - Concurrent processing of multiple neurons within one layer
    - This involves matrix multiplication
        - Matrix multiplication is used to calculate outputs based on weights and biases
- **Hybrid Parallelism:**
  A mix of two or more parallel training methods (3D parallelism uses all three)
    - Examples of uses of hybrid parallelism are Megatron-LM and DDP
    - Megatron-LM (Shoeybi et al.)
        - Uses Data and Model parallelism
        - Data parallelism: training mini-batch split among multiple workers
            - Remember how batches are a subset of data
        - Model parallelism: memory usage and computation is split among multiple workers

Works Cited

Shoeybi, Mohammad, et al. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism, 13 Mar. 2020, arxiv.org/pdf/1909.08053.