CS 205: Parallelized Hyperparameter Tuning

Roy Han, Evan Jiang

February 20, 2023

Our project is aimed at addressing the problem of massively parallelizing hyperparameter optimization. In machine learning, hyperparameters are controlled model variables that are set before and unaffected by training. As one would expect, the selection of hyperparameters is critical to the performance of the model, and tuning them appropriately is an important facet of training.

We look to Asynchronous Successive Halving (ASH) for motivations on massively parallelizing hyperparameter optimization. ASH focuses on maximizing resource utilization by distributing candidate models amongst computing resources, and repeatedly redistributing the top performing models amongst those resources. The computational bottleneck comes from finding the optimal hyperparameter configuration – to do so, people usually have to train a bunch of models on a dataset and compare them with one another to determine which is the best. However, this training is typically done sequentially (one set of hyperparameters at a time), and that process is time-intensive and expensive.

We believe that ASH can parallelize this process via a MIMD (multiple instruction multiple data) architecture, as we aim to leverage multiple cores to be training models using multiple datastreams (different hyperparameter configurations). This requires us to be working with distributed memory models (to coordinate which sets of configurations to advance to the next rung), as well as dynamically allocating new jobs to available processors. To this end, the type of parallelism being leveraged will be task-based, as training of individual models with certain hyperparameters will be the unit in which distribution is conducted.

While the amount of memory needed to train a model can vary, large datasets with complex models can require multiple GB of memory; since each node capable of 64GB of memory, and we'll be likely working with $4^4 = 256$ possible hyperparameter configurations on a reasonably-sized dataset, distributing this task for different hyperparameters among such nodes is indeed suitable, with $(256+128+64+32+16+8+4+2+1) = 511$ epochs (50GB memory estimate). We predict that each epoch should take less than a minute to train, so the entire process should take roughly $511/64 \approx 8$ min total (with 64 processors). This provides a roughly accurate estimate, since we do not need to conduct any synchronization at each rung of halving since we asynchronously promote candidate models to the next rung.