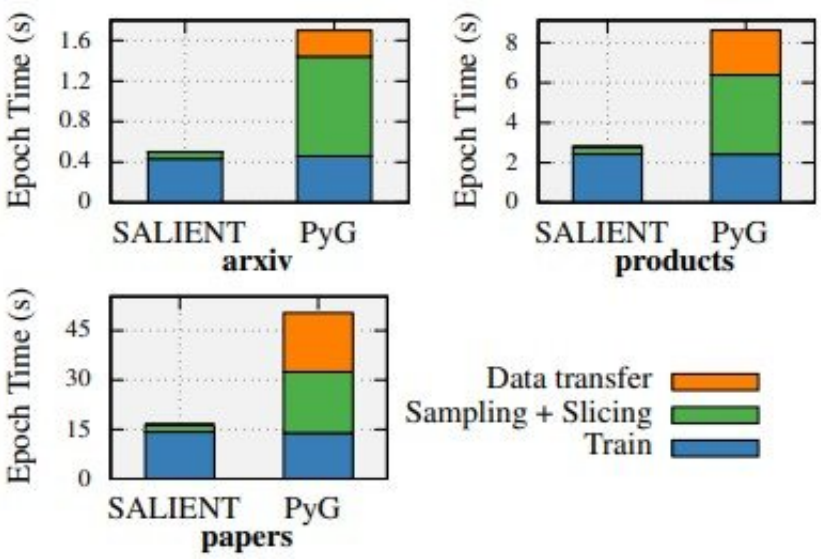
Sampling and pipelining method speeds up deep learning on large graphs

November 30 2022, by Lauren Hinkel



Performance improvement of SALIENT over standard PyG workflow. Timing measurements on one machine with one GPU. GNN: GraphSAGE with fanout (15, 10, 5). Credit: *arXiv* (2021). DOI: 10.48550/arxiv.2110.08450

Graphs, a potentially extensive web of nodes connected by edges, can be used to express and interrogate relationships between data, like social connections, financial transactions, traffic, energy grids, and molecular interactions. As researchers collect more data and build out these graphical pictures, researchers will need faster and more efficient methods, as well as more computational power, to conduct deep learning on them, in the way of graph neural networks (GNN).

Now, a new method, called SALIENT (SAmpling, sLIcing, and data movemeNT), developed by researchers at MIT and IBM Research, improves the training and inference performance by addressing three key bottlenecks in computation. This dramatically cuts down on the runtime of GNNs on large datasets, which, for example, contain on the scale of 100 million nodes and 1 billion edges. Further, the team found that the technique scales well when computational power is added from one to 16 graphical processing units (GPUs). The work was presented at the Fifth Conference on Machine Learning and Systems.

"We started to look at the challenges current systems experienced when scaling state-of-the-art machine learning techniques for graphs to really big datasets. It turned out there was a lot of work to be done, because a lot of the existing systems were achieving good performance primarily on smaller datasets that fit into GPU memory," says Tim Kaler, the lead author and a postdoc in the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL).

By vast datasets, experts mean scales like the entire Bitcoin network, where certain patterns and data relationships could spell out trends or foul play. "There are nearly a billion Bitcoin transactions on the blockchain, and if we want to identify illicit activities inside such a joint network, then we are facing a graph of such a scale," says co-author Jie Chen, senior research scientist and manager of IBM Research and the MIT-IBM Watson AI Lab. "We want to build a system that is able to handle that kind of graph and allows processing to be as efficient as possible, because every day we want to keep up with the pace of the new data that are generated."

Kaler and Chen's co-authors include Nickolas Stathas MEng '21 of Jump Trading, who developed SALIENT as part of his graduate work; former MIT-IBM Watson AI Lab intern and MIT graduate student Anne Ouyang; MIT CSAIL postdoc Alexandros-Stavros Iliopoulos; MIT

CSAIL Research Scientist Tao B. Schardl; and Charles E. Leiserson, the Edwin Sibley Webster Professor of Electrical Engineering at MIT and a researcher with the MIT-IBM Watson AI Lab.

For this problem, the team took a systems-oriented approach in developing their method: SALIENT, says Kaler. To do this, the researchers implemented what they saw as important, basic optimizations of components that fit into existing machine-learning frameworks, such as PyTorch Geometric and the deep graph library (DGL), which are interfaces for building a machine-learning model.

Stathas says the process is like swapping out engines to build a faster car. Their method was designed to fit into existing GNN architectures, so that domain experts could easily apply this work to their specified fields to expedite model training and tease out insights during inference faster. The trick, the team determined, was to keep all of the hardware (CPUs, data links, and GPUs) busy at all times: while the CPU samples the graph and prepares mini-batches of data that will then be transferred through the data link, the more critical GPU is working to train the machine-learning model or conduct inference.

The researchers began by analyzing the performance of a commonly used machine-learning library for GNNs (PyTorch Geometric), which showed a startlingly low utilization of available GPU resources. Applying simple optimizations, the researchers improved GPU utilization from 10 to 30%, resulting in a 1.4 to two times performance improvement relative to public benchmark codes. This fast baseline code could execute one complete pass over a large training dataset through the algorithm (an epoch) in 50.4 seconds.

Seeking further performance improvements, the researchers set out to examine the bottlenecks that occur at the beginning of the data pipeline: the algorithms for graph sampling and mini-batch preparation. Unlike

other neural networks, GNNs perform a neighborhood aggregation operation, which computes information about a node using information present in other nearby nodes in the graph—for example, in a social network graph, information from friends of friends of a user.

As the number of layers in the GNN increase, the number of nodes the network has to reach out to for information can explode, exceeding the limits of a computer. Neighborhood sampling algorithms help by selecting a smaller random subset of nodes to gather; however, the researchers found that current implementations of this were too slow to keep up with the processing speed of modern GPUs.

In response, they identified a mix of data structures, algorithmic optimizations, and so forth that improved sampling speed, ultimately improving the sampling operation alone by about three times, taking the per-epoch runtime from 50.4 to 34.6 seconds. They also found that sampling, at an appropriate rate, can be done during inference, improving overall energy efficiency and performance, a point that had been overlooked in the literature, the team notes.

In previous systems, this sampling step was a multi-process approach, creating extra data and unnecessary data movement between the processes. The researchers made their SALIENT method more nimble by creating a single process with lightweight threads that kept the data on the CPU in shared memory. Further, SALIENT takes advantage of a cache of modern processors, says Stathas, parallelizing feature slicing, which extracts relevant information from nodes of interest and their surrounding neighbors and edges, within the shared memory of the CPU core cache. This again reduced the overall per-epoch runtime from 34.6 to 27.8 seconds.

The last bottleneck the researchers addressed was to pipeline mini-batch data transfers between the CPU and GPU using a prefetching step,

which would prepare data just before it's needed. The team calculated that this would maximize bandwidth usage in the data link and bring the method up to perfect utilization; however, they only saw around 90%. They identified and fixed a performance bug in a popular PyTorch library that caused unnecessary round-trip communications between the CPU and GPU. With this bug fixed, the team achieved a 16.5 second per-epoch runtime with SALIENT.

"Our work showed, I think, that the devil is in the details," says Kaler. "When you pay close attention to the details that impact performance when training a graph neural network, you can resolve a huge number of performance issues. With our solutions, we ended up being completely bottlenecked by GPU computation, which is the ideal goal of such a system."

SALIENT's speed was evaluated on three standard datasets ogbn-arxiv, ogbn-products, and ogbn-papers100M, as well as in multi-machine settings, with different levels of fanout (amount of data that the CPU would prepare for the GPU), and across several architectures, including the most recent state-of-the-art one, GraphSAGE-RI. In each setting, SALIENT outperformed PyTorch Geometric, most notably on the large ogbn-papers100M dataset, containing 100 million nodes and over a billion edges Here, it was three times faster, running on one GPU, than the optimized baseline that was originally created for this work; with 16 GPUs, SALIENT was an additional eight times faster.

While other systems had slightly different hardware and experimental setups, so it wasn't always a direct comparison, SALIENT still outperformed them. Among systems that achieved similar accuracy, representative performance numbers include 99 seconds using one GPU and 32 CPUs, and 13 seconds using 1,536 CPUs. In contrast, SALIENT's runtime using one GPU and 20 CPUs was 16.5 seconds and was just two seconds with 16 GPUs and 320 CPUs.

"If you look at the bottom-line numbers that prior work reports, our 16 GPU runtime (two seconds) is an order of magnitude faster than other numbers that have been reported previously on this dataset," says Kaler.

The researchers attributed their performance improvements, in part, to their approach of optimizing their code for a single machine before moving to the distributed setting. Stathas says that the lesson here is that for your money, "it makes more sense to use the hardware you have efficiently, and to its extreme, before you start scaling up to multiple computers," which can provide significant savings on cost and carbon emissions that can come with model training.

This new capacity will now allow researchers to tackle and dig deeper into bigger and bigger graphs. For example, the Bitcoin network that was mentioned earlier contained 100,000 nodes; the SALIENT system can capably handle a graph 1,000 times (or three orders of magnitude) larger.

"In the future, we would be looking at not just running this graph neural network training system on the existing algorithms that we implemented for classifying or predicting the properties of each node, but we also want to do more in-depth tasks, such as identifying common patterns in a graph (subgraph patterns), [which] may be actually interesting for indicating financial crimes," says Chen.

"We also want to identify nodes in a graph that are similar in a sense that they possibly would be corresponding to the same bad actor in a financial crime. These tasks would require developing additional algorithms, and possibly also neural network architectures."

More information: Tim Kaler et al, Accelerating Training and Inference of Graph Neural Networks with Fast Sampling and Pipelining, *arXiv* (2021). DOI: 10.48550/arxiv.2110.08450