Problem Set 4: Hyperparameter Optimization

CS4787/5777 — Principles of Large-Scale ML Systems

This problem set is assigned on November 13, 2022 and is due about two weeks later on December 4, 2022 at 11:59PM. Please do not wait until the last minute to do this assignment.

Problem 1: Parallelism and Parallel Hardware. In class, we talked about how parallelism can be used to speed up a program. In this problem, we'll explore some aspects of parallelism.

1(a). Suppose that you want to use parallelism (specifically, multi-core parallelism) to speed up a program you have. You estimate that about 1% of the runtime of your original non-parallel program is spent performing non-parallelizable tasks, and the rest of your program is parallelizable. What speedup does Amdahl's law predict for your program if you run it on 4 CPU cores? What if you run it on 256 distributed CPU cores?

1(b). For each of the following scenarios, describe which (if any) of the following source(s) of hardware parallelism on CPUs, if any, would tend to be useful to speed up the task at the level of abstraction described, from among the following options: SIMD, Multicore, Distributed. Explain your answer.

(i) Aragorn wants to run grid search for hyperparameter optimization.

(ii) Bill wants to use parallelism to speed up an already-written sequential application written in FORTRAN in the 1970s, without having to change the code or explicitly invoke parallelism.

(iii) Celeborn wants to acquire training data for a digital agriculture task by personally going out to many fields and photographing cow herds.

(iv) Dwalin wants to deploy a deep neural network to correct misspellings on his customers' Android phones.

(v) Elrond wants to compute $\mathrm{ReLU}(x)$ for a small vector $x \in \mathbb{R}^{64}$ and store the result in an array.

(vi) Frodo wants to preprocess a large number (millions) of documents using a series of regular expressions to convert sentences to feature vectors for an NLP application.

(vii) Gimli wants to sort a large number (millions) of emails into either spam or not-spam categories using a support vector machine he's already trained.

(viii) Harry wants to multiply two floating-point numbers.

(Note that we are not looking for any specific answer here, but rather a reasonable answer with a good short justification!)

1(c). Your friend Isildur claims that he observed a $9\times$ speedup from rewriting his code to be distributed over 8 workers. Is Isildur's claimed speedup consistent with what we would expect vectorization to be able to achieve on a standard Intel or AMD CPU? Why or why not?

1(d). For each of the following computational tasks, say whether you think it would help speed it up to run it on a GPU instead of a CPU, and explain why. Also, for each one where GPU acceleration would be possible, indicate whether you could accomplish the task using a single operation from the CUDA cuBLAS linear algebra library,[1]. If you could use a cuBLAS operation, say which one.

(i) Kili wants to compute a 8192×8192 by 8192×8192 matrix-matrix multiply.

(ii) Legolas wants to compile a C++ program, producing an optimized executable.

(iii) Merry wants to compute the operation $y = \alpha \cdot Ax + \beta y$ for some vectors x and y, matrix A, and scalars α and β.

(iv) Pippin wants to wants to render 3d computer graphics images for a video game.

(v) Sam wants to compute the gradient of the loss for a Transformer network via backpropagation.

(Note again that we are not necessarily looking for a specific answer here, but rather a reasonable answer with a good short justification!)

[1] Documentation for cuBLAS can be found online at `https://docs.nvidia.com/cuda/cublas/index.html`

Problem 2: Memory. In this problem, we will examine the memory usage of the learning algorithms we've discussed in class.

Suppose that we want to run empirical risk minimization on a simple multinomial logistic regression task, such as in PA2. Recall that the parameters are a matrix $W \in \mathbb{R}^{c \times d}$, and we wanted to minimize

$$f(W) = \sum_{i=1}^{n} \ell(W; x_i, y_i)$$

where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}^c$ are the ith training example and label respectively, and ℓ is defined as appropriate for a component loss for multinomial logistic regression.

Suppose further that (as in the programming assignments) we want to run this on the MNIST dataset, where $d = 28 \times 28$, $c = 10$, and $n = 60000$. Also assume that we are running on a CPU with a 32 KiB L1 cache, a 2 MiB L2 cache, and an 32 MiB L3 cache.[2]

2(a). Suppose that single-precision floating point numbers are used for all the numbers in the algorithm. How many bytes of memory are needed to store each of the following? Also, for each of the following, what is the smallest cache in which it data will fit (assuming it is the only thing being stored in cache)?

i. a single copy of the parameter vector W

ii. a single training example x_i

iii. the entire training set $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$.

2(b). When optimizing the memory performance of an algorithm, one thing that is often important in practice that I did not mention in class is to *avoid memory allocation*. Allocating new memory on each iteration of SGD to store the results of numerical operations (such as matrix multiplies) can have substantial overhead when compared with the performance of a system that always stores those results in a pre-allocated array. This can be especially useful on an accelerator such as a GPU.

Recall that the update loop of SGD on this problem is to sample a random training set index i_t uniformly from $\{1, \ldots, n\}$ and then run.

$$W_{t+1} = W_t - \alpha_t \cdot (\mathrm{softmax}(W_t x_{i_t}) - y_{i_t}) x_{i_t}^T.$$

Write out pseudocode for an implementation of SGD that would allocate no memory in this inner loop (any memory needed must be allocated *once* at the start of the algorithm).

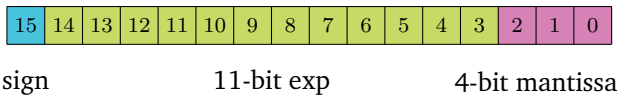
[2] This roughly corresponds to an Intel Raptor Lake CPU.

Algorithm 1 Preallocated SGD

input: Data x, y, learning rate schedule α_t, initial parameters w_0, number of iterations T.
initialize $W \in \mathbb{R}^{c \times d} \leftarrow w_0$
initialize $B \leftarrow \mathbf{0}^{c+2}$. Entries 0 through $c-1$ are class buffers, entry c is for the sum, and entry $c+1$ is a multiplication buffer.
initialize $i_t \leftarrow 0$
for $t = 0$ **to** $T-1$ **do**
  $i_t \leftarrow \text{UNIFORMINT}(0, n-1)$ assuming UNIFORMINT runs in place. This can practically be done by reading from a source like `/dev/urandom` and then taking mod w/rt to n.
  for $i = 0$ **to** $c-1$ **do**
    for $j = 0$ **to** $d-1$ **do**
      $B_{c+1} \leftarrow W_{i,j} X_{i_t j}$
      $B_i \leftarrow B_i + B_{c+1}$
    end for
    $B_i \leftarrow e^{B_i}$
    $B_c \leftarrow B_c + B_i$
  end for
  for $i = 0$ **to** $c-1$ **do**
    $B_i \leftarrow \frac{B_i}{B_c}$
    $B_i \leftarrow B_i - Y_{i_t i}$
    $B_i \leftarrow \alpha_t B_i$
  end for
  for $i = 0$ **to** $c-1$ **do**
    for $j = 0$ **to** $d-1$ **do**
      $B_{c+1} \leftarrow B_i X_{i_t j}$
      $W_{i,j} \leftarrow W_{i,j} - B_{c+1}$
    end for
  end for
end for
return W

2(c). Apart from memory used to store the input training examples, how much memory does your algorithm from part 1(b) use? What is the smallest cache that all that data could fit in?

Problem 3: Low-Precision Arithmetic. Your friend Tom has invented his own 16-bit floating point format, called TomFloat. Tom's format has even more exponent bits than either standard 16-bit floats or bfloats. Specifically, it has 1 sign bit, 12 exponent bits, and 3 mantissa bits, as illustrated below.



Among (1) IEEE standard 16-bit floats, (2) bfloat, (3) TomFloat, (4) IEEE standard 32-bit floats, and (5) IEEE standard 64-bit floats:

3(a). How are they ordered in terms of their machine epsilon? (That is, which has the largest machine epsilon? Which has the smallest? How are the others ordered relative to each other in between?)

3(b). How are they ordered in terms of their numerical range? (That is, which has the largest numerical range? Which has the smallest? How are the others ordered relative to each other in between?)

3(c). Which do we expect would result in the greatest quantization error for an ML application? *Your answer here may be subjective.*

Briefly justify each of your answers.