K Nearest Neighbor Design

Gideon Lee
NumberSciences

Overall Objective

The objective of the K-nearest neighbor optimizer is to find a pair of distance function and weighting function that minimizes the error.

1. The distance defines the distance between two data samples, D(X, Y).
2. The distance function is used to identify the K nearest samples of X known as X1, X2, … XK.
3. Each sample is associated with an effect E(X).
4. The effects of the K nearest samples of Y are used together to make a predicted effect for Y through a weighting function W(E(X1), E(X2), … E(XK)).
5. The error of the predicted effect and the actual effect of Y is then measured.
6. The aggregate error of all data samples {Y} is summed and returned.
7. The goal is to find D() and W() that give the lowest aggregate error.

GPU implementation outline

To take advantage of the GPU parallel computing power, we make some simplifying assumptions in the implementation of steps 1 to 6.

1. The distance function perceives each data sample through an ensemble of technical indicators T(). Given a data sequence leading up to X, a technical indicator computes a scalar T(X) value.
2. The scalar technical value is then mapped to a ranked value in the range of 0 to 127, R(X) by a precomputed statistical look up that maps the T(X) range for each of 0 to 127.
3. Once mapped, a sample value X has the semantic form of (T1(X), T2(X), … T16(X)). It can be presented using a 16 byte block. The effect of X is represented as the upper most bits. Its value can therefore range from 0 to 65535.
4. The distance function D(X, Y) takes two such 16 byte blocks to compute a distance value.
5. For each point Y, the distance and the effect of the K nearest (by default 16 nearest) samples are tracked. The distance is 16-bit. The effect is 16-bit. It is therefore a 32-bit pair. For example, to represent 16 nearest samples, 4 byte * 16 = 64 bytes are necessary.
6. A worker thread is dedicated for each sample. Register-level private memory is used to represent the sample Y and the K nearest X neighbors.
7. On-chip shared memory (also called __local memory) is used for buffering the batch of data samples. That way, all the worker threads will read the same data sample X together and compute the D(X, Y) together. The insertion also happens in parallel by running the D(X) against all the K nearest neighbors up to that point.
8. Due to the number of worker threads C that can be available given the 64 bytes minimum register size, only C number of samples Y can be processed in each sub-pass. After a pass, the predicted effect of Y is computed based on its K nearest neighbors based on W(E(X1), E(X2), … E(XK)). The actual effect of Y is then used to compute the error. The aggregate error of W() is computed at the end of each sub-pass.
9. Repeated calls of the sub-passes are made so that the aggregate errors for all the predictions is computed.
10. The outer loop is responsible for optimizing the parameters used by D() and W().

If possible, steps 4-9 are performed in one GPU enqueue invocation. If the data sample size is too big,
it may be broken out in sub-chunks. Even if the data sample size fits all in the GPU memory, it may be necessary to break up the sub-pass calculation in multiple enqueu invocations because of the maximum wait time is afforded for a GPU kernel invocation.

**CPU Implementation outline**

The steps 1 to 3 and step 10 in the above implementation outline are actually carried out by the CPU. Here are some useful points.

1. Step (1) can actually be computed using a different GPU kernel. It is often done in a “bar parallel” manner. The original data samples are processed in parallel by a technical indicator $T(X)$. Then, step (2) is carried out by the data samples are compared in parallel to the ranking map table (which is a constant) to arrive at a value of 0 to 127.

2. Since many rounds of optimization may be necessary for fine tuning the parameters of $D()$ and $W()$, the time required to do step (1) and step (2) is negligible. It is therefore not a huge disadvantage if step (1) is done by the CPU.

3. The $W()$ function should generally be declining in weight along increasing distance.

4. The $D()$ function assign different weights to different technical indicators.

5. The choice of the technical indicators $T1()$ … $T16()$ and the optimization of parameters used by the technical indicators is done in the outermost loop.

**Overall Looping**

The overall nesting can be seen this way:

- Choice of technical indicators and their parameters (outermost loop, probably different CPU/GPU)
  - Optimization of the parameters of $D()$ and $W()$ (CPU loop)
    - Sub-passes given $D()$ and $W()$ (GPU loop, preferable invoked in one enqueue)
      - Sub-Pass (GPU loop)
        - Batches (GPU loop) each batch loads the maximum number of samples for __local
          - Batch (GPU parallel threads)

**Conclusion**

The K-nearest neighbor optimizer provides the triage step of statistical learning. The output is not the easiest to reapply since it requires a full data set. However, the output can be used to identify the set of technical indicators that can be subjected to other more traditional statistical learning algorithms.

The time required to compute the inner loops makes the time required to compute a single technical indicator largely an irrelevant issue. While it may be beneficial to do the technical indicator calculation using GPU, the performance benefit may be negligible for optimization purpose. On the other hand, it may be useful for real time computation.