Kozachenko and Leonenko Entropy Estimation with the help of GPUs

Paul Ivanov

UC Berkeley
Vision Science Graduate Group

Redwood Center for Theoretical Neuroscience
http://redwood.berkeley.edu/

March 7, 2008
Overview of Algorithm

Divide set into Targets and Neighbors (randomly)
randomize order of Neighbors
for (L = 0; L < levels; L++)
    for each $t \in$ Targets
        look through the first $N = 2^L$ Neighbors
        and find the closest (at distance $D \times N$)

$T \sum_{t=1}^{T} \log D \times N$, $t = \text{avg log NN distance (from } N \text{ samples)}$
proximity distribution function

We will use it to estimate $-\log f_X(x)$

Ivanov (Redwood Center)
Overview of Algorithm

Divide set into Targets and Neighbors (randomly)

for (\(L = 0; L < \text{levels}; L++\))
  for each \(t \in \text{Targets} \)
    look through the first \(N = 2^L\) Neighbors
    and find the closest (at distance \(D \ast N, t\))

\[ \sum_{t=1}^{T} \log D \ast N, t = \text{avg log NN distance (from } N \text{ samples)} \]

proximity distribution function

avg log \(D \ast (\text{above})\) as a function of \(N\)

We will use it to estimate \(-\log f_X(x)\)
Overview of Algorithm

Divide set into **Targets** and **Neighbors** (randomly)

randomize order of Neighbors

for (L = 0; L < levels; L++)

for each $t \in$ Targets

look through the first $N = 2^L$ Neighbors

and find the closest (at distance $D \ast N$, $t$)

$\sum_{t=1}^{T} \log D \ast N, t = \text{avg log NN distance (from } N \text{ samples)}$

proximity distribution function

avg log $D \ast (\text{above})$ as a function of $N$

We will use it to estimate $-\log f_X(x)$

Ivanov (Redwood Center)
Overview of Algorithm

Divide set into Targets and Neighbors (randomly)

randomize order of Neighbors
for (L = 0; L < levels; L++)

Look through the first \( N = 2^L \) Neighbors and find the closest (at distance \( D \cdot N \), \( t \))

\[
T \sum_{t=1}^{T} \log D \cdot N, t = \text{avg log NN distance (from} \ N \text{samples)}
\]

proximity distribution function

avg log \( D \cdot N \) (above) as a function of \( N \)

We will use it to estimate \(-\log 2 f_X(x)\)
Overview of Algorithm

Divide set into **Targets** and **Neighbors** (randomly)

randomize order of Neighbors

for \((L = 0; \; L < \text{levels}; \; L++)\)

  for each \(t \in \text{Targets}\)

\[
\sum_{t=1}^{T} \log D \cdot N, \; t = \text{avg log NN distance (from} \; N \; \text{samples)}
\]

proximity distribution function

avg log \(D \cdot N\) (above) as a function of \(N\)

We will use it to estimate \(-\log_2 f_X(x)\)
Overview of Algorithm

Divide set into Targets and Neighbors (randomly)

randomize order of Neighbors
for (L = 0; L < levels; L++)
    for each $t \in$ Targets
        look through the first $N = 2^L$ Neighbors

proximity distribution function

We will use it to estimate $-\log_2 f_X(x)$
Overview of Algorithm

Divide set into **Targets** and **Neighbors** (randomly)

- randomize order of Neighbors
- for (L = 0; L < levels; L++)
  - for each \( t \in \text{Targets} \)
    - look through the first \( N = 2^L \) Neighbors
    - and find the closest (at distance \( D_{N,t}^* \))
Overview of Algorithm

Divide set into **Targets** and **Neighbors** (randomly)

randomize order of Neighbors

for (L = 0; L < levels; L++)
  for each $t \in$ Targets
    look through the first $N = 2^L$ Neighbors
    and find the closest (at distance $D_{N,t}^*$)

$$\frac{1}{T} \sum_{t=1}^{T} \log D_{N,t}^* = \text{avg log NN distance (from } N \text{ samples)}$$
Overview of Algorithm

Divide set into **Targets** and **Neighbors** (randomly)

randomize order of Neighbors

for (L = 0; L < levels; L++)
    for each \( t \in \text{Targets} \)
        look through the first \( N = 2^L \) Neighbors
        and find the closest (at distance \( D_{N,t}^* \))

\[
\frac{1}{T} \sum_{t=1}^{T} \log D_{N,t}^* = \text{avg log NN distance (from } N \text{ samples)}
\]

**proximity distribution function**

avg log \( D^* \) (above) as a function of \( N \)
Overview of Algorithm

Divide set into **Targets** and **Neighbors** (randomly)

- randomize order of Neighbors
- for \( L = 0; L < \text{levels}; L++ \)
  - for each \( t \in \text{Targets} \)
    - look through the first \( N = 2^L \) Neighbors
    - and find the closest (at distance \( D^*_{N,t} \))

\[
\frac{1}{T} \sum_{t=1}^{T} \log D^*_{N,t} = \text{avg log NN distance (from } N \text{ samples)}
\]

**proximity distribution** function

- avg log \( D^* \) (above) as a function of \( N \)
- We will use it to estimate \(-\log_2 f_X(x)\)
An Example

Goal

How much disorder there is in the distribution of people living in California. (How many bits do we need to represent their location?)
### An Example

#### Goal

How much disorder there is in the distribution of people living in California. (How many bits do we need to represent their location?)

<table>
<thead>
<tr>
<th>City</th>
<th>fresno</th>
<th>+san diego</th>
<th>+backersfield stockton</th>
<th>+davis, san francisco riverside, santa cruz</th>
</tr>
</thead>
<tbody>
<tr>
<td>berkeley</td>
<td>155</td>
<td>155</td>
<td>49</td>
<td>12</td>
</tr>
<tr>
<td>mt view</td>
<td>135</td>
<td>135</td>
<td>55</td>
<td>33</td>
</tr>
<tr>
<td>santa monica</td>
<td>215</td>
<td>119</td>
<td>119</td>
<td>65</td>
</tr>
<tr>
<td>roseville</td>
<td>160</td>
<td>160</td>
<td>56</td>
<td>28</td>
</tr>
<tr>
<td>avg log $D^*$</td>
<td>2.214</td>
<td>2.150</td>
<td>1.813</td>
<td>1.464</td>
</tr>
</tbody>
</table>
70% of CA lives in SF Bay and LA Metro areas
\[ h(X) \triangleq -\int_{x \in \mathcal{X}} f_X(x) \log_2 f_X(x) \, dx \]

\[ = \int_{x \in \mathcal{X}} f_X(x) i_X(x) \, dx \]

\[ = E \{ i_X(x) \} \]

\[ \approx \frac{1}{M} \sum_{m=1}^{M} i_X(x_m) \]

Assuming \( f_X(x) \, dx \approx \frac{1}{M}, \forall x_m; \)

\[ \hat{i}_X(x) \triangleq -\log_2 f_X(x) \]

\[ \hat{i}_X(x) = k E \{ \log_2 D_N^k \} + \log_2 \left( \frac{A_k N}{k} \right) + \frac{\gamma}{\ln 2} \]
\[ h(X) \triangleq -\int_{x \in \mathcal{X}} f_X(x) \log_2 f_X(x) \, dx \]

\[ = \int_{x \in \mathcal{X}} f_X(x) \hat{i}_X(x) \, dx \]

\[ = E \{ \hat{i}_X(x) \} \]

\[ \approx \frac{1}{M} \sum_{m=1}^{M} \hat{i}_X(x_m) \]

\[ \hat{i}_X(x) \triangleq -\log_2 f_X(x) \]

assuming \( f_X(x) \, dx \approx \frac{1}{M}, \forall x_m; \)

\[ \hat{i}_X(x) = k E \{ \log_2 D_N^* \} + \log_2 \left( \frac{A_k N}{k} \right) + \frac{\gamma}{\ln 2} \]

\[ E \{ \log_2 D_N^* \} \approx \frac{1}{T} \sum_{t=1}^{T} \log_2 D_{N,t}^* \]
Theory

\[ h(X) \triangleq -\int_{x \in \mathcal{X}} f_X(x) \log_2 f_X(x) \, dx \]
\[ = \int_{x \in \mathcal{X}} f_X(x) i_X(x) \, dx \]
\[ = E \{ i_X(x) \} \]
\[ \approx \frac{1}{M} \sum_{m=1}^{M} \hat{i}_X(x_m) \]

\[ i_X(x) \triangleq -\log_2 f_X(x) \]

assuming \( f_X(x) \, dx \approx \frac{1}{M}, \forall x_m; \)

\[ \hat{i}_X(x) = kE \{ \log_2 D_{N}^{*} \} + \log_2 \left( \frac{A_k N}{k} \right) + \frac{\gamma}{\ln 2} \]

\[ E \{ \log_2 D_{N}^{*} \} \approx \frac{1}{T} \sum_{t=1}^{T} \log_2 D_{N,t}^{*}. \]

\[ h(X) \approx \frac{k}{M} \sum_{m=1}^{M} \log_2 D_{N,m}^{*} + \log_2 \left( \frac{A_k N}{k} \right) + \frac{\gamma}{\ln 2} \]

\[ A_k = k \pi^{k/2} / \Gamma \left( \frac{k}{2} + 1 \right) \]
1. Create library of neighbor patches

\[ \{X^{(A_n)}\} = \begin{array}{cccc}
\text{patch} & \text{patch} & \text{patch} & \ldots \\
\end{array} \]

2. Pick a target patch

\[ X^{(A_t)} = \begin{array}{c}
\text{target patch}
\end{array} \]

3. Calculate Euclidean distance between target and nearest neighbor as a function of number of neighbors

\[ \min_n \| X^{(A_t)} - X^{(A_n)} \|_{L^2} \]
GPGPU with CUDA

http://courses.ece.uiuc.edu/ece498AL1
David Kirk - NVIDIA and Wen-mei W. Hwu - UIUC
GeForce 8800

16 highly threaded SM’s, >128 FPU’s, 367 GFLOPS, 768 MB DRAM, 86.4 GB/S Mem BW, 4GB/S BW to CPU
Why Massively Parallel Processor

• A quiet revolution and potential build-up
  – Calculation: 367 GFLOPS vs. 32 GFLOPS
  – Memory Bandwidth: 86.4 GB/s vs. 8.4 GB/s
  – Until last year, programmed through graphics API

GPU in every PC and workstation – massive volume and potential impact

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007
ECE 498AL1, University of Illinois, Urbana-Champaign
What is Behind such an Evolution?

• The GPU is specialized for compute-intensive, highly data parallel computation (exactly what graphics rendering is about)
  - So, more transistors can be devoted to data processing rather than data caching and flow control

• The fast-growing video game industry exerts strong economic pressure that forces constant innovation
## Previous Projects

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>Source</th>
<th>Kernel</th>
<th>% time</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264</td>
<td>SPEC ’06 version, change in guess vector</td>
<td>34,811</td>
<td>194</td>
<td>35%</td>
</tr>
<tr>
<td>LBM</td>
<td>SPEC ’06 version, change to single precision and print fewer reports</td>
<td>1,481</td>
<td>285</td>
<td>&gt;99%</td>
</tr>
<tr>
<td>RC5-72</td>
<td>Distributed.net RC5-72 challenge client code</td>
<td>1,979</td>
<td>218</td>
<td>&gt;99%</td>
</tr>
<tr>
<td>FEM</td>
<td>Finite element modeling, simulation of 3D graded materials</td>
<td>1,874</td>
<td>146</td>
<td>99%</td>
</tr>
<tr>
<td>RPES</td>
<td>Rye Polynomial Equation Solver, quantum chem, 2-electron repulsion</td>
<td>1,104</td>
<td>281</td>
<td>99%</td>
</tr>
<tr>
<td>PNS</td>
<td>Petri Net simulation of a distributed system</td>
<td>322</td>
<td>160</td>
<td>&gt;99%</td>
</tr>
<tr>
<td>SAXPY</td>
<td>Single-precision implementation of saxpy, used in Linpack’s Gaussian elim. routine</td>
<td>952</td>
<td>31</td>
<td>&gt;99%</td>
</tr>
<tr>
<td>TRACF</td>
<td>Two Point Angular Correlation Function</td>
<td>536</td>
<td>98</td>
<td>96%</td>
</tr>
<tr>
<td>FDTD</td>
<td>Finite-Difference Time Domain analysis of 2D electromagnetic wave propagation</td>
<td>1,365</td>
<td>93</td>
<td>16%</td>
</tr>
<tr>
<td>MRI-Q</td>
<td>Computing a matrix Q, a scanner’s configuration in MRI reconstruction</td>
<td>490</td>
<td>33</td>
<td>&gt;99%</td>
</tr>
</tbody>
</table>
Speedup of Applications

- GeForce 8800 GTX vs. 2.2GHz Opteron 248
- 10× speedup in a kernel is typical, as long as the kernel can occupy enough parallel threads
- 25× to 400× speedup if the function’s data requirements and control flow suit the GPU and the application is optimized
- Keep in mind that the speedup also reflects how suitable the CPU is for executing the kernel
What is the GPU Good at?

• The GPU is good at data-parallel processing
  • The same computation executed on many data elements in parallel – low control flow overhead with high SP floating point arithmetic intensity
  • Many calculations per memory access
  • Currently also need high floating point to integer ratio

• High floating-point arithmetic intensity and many data elements mean that memory access latency can be hidden with calculations instead of big data caches – Still need to avoid bandwidth saturation!
A Common Programming Pattern

• Local and global memory reside in device memory (DRAM) - much slower access than shared memory
• So, a profitable way of performing computation on the device is to **block data** to take advantage of fast shared memory:
  – **Partition** data into **data subsets** that fit into shared memory
  – **Handle each data subset with one thread block** by:
    • Loading the subset from global memory to shared memory, **using multiple threads to exploit memory-level parallelism**
    • Performing the computation on the subset from shared memory; each thread can efficiently multi-pass over any data element
    • Copying results from shared memory to global memory
Thread Batching: Grids and Blocks

- A kernel is executed as a grid of thread blocks
  - All threads share data memory space
- A thread block is a batch of threads that can cooperate with each other by:
  - Synchronizing their execution
    - For hazard-free shared memory accesses
  - Efficiently sharing data through a low latency shared memory
- Two threads from two different blocks cannot cooperate
Divide set into **Targets** and **Neighbors** (randomly)

randomize order of Neighbors

for $(L = 0; L < \text{levels}; L++)$

   for each $t \in \text{Targets}$

      look through the first $N = 2^L$ Neighbors

      and find the closest (at distance $D_{N,t}^*$)

$$\frac{1}{T} \sum_{t=1}^{T} \log D_{N,t}^* = \text{avg log NN distance (from } N \text{ samples)}$$

**proximity distribution function**

avg log $D^*$ (above) as a function of $N$

We will use it to estimate $-\log_2 f_X(x)$
Back to the Kozachenko Leonenko Estimator

Divide set into **Targets** and **Neighbors** (randomly)

randomize order of Neighbors

for \((L = 0; L < \text{levels}; L++)\)

\hspace{1cm} \text{for each } t \in \text{Targets} - \text{Parallelize here}

\hspace{1cm} \text{look through the first } N = 2^L \text{ Neighbors- here}

\hspace{1cm} \text{and find the closest (at distance } D_{N,t}^*\text{)- and maybe here}

\[ \frac{1}{T} \sum_{t=1}^{T} \log D_{N,t}^* = \text{avg log NN distance (from } N \text{ samples)} \]

**proximity distribution function**

avg log \( D^* \) (above) as a function of \( N \)

We will use it to estimate \(- \log_2 f_X(x)\)


David Kirk - NVIDIA and Wen-mei W. Hwu - UIUC
http://courses.ece.uiuc.edu/ece498AL1

Thank you.