Self Organizing Maps on the GPU

Mark Kim
University of Utah
Scientific Computing and Imaging Institute
Los Alamos National Laboratory
Summer 2009

Organization

- What's a SOM?
- How's it used?
- Creating a SOM
- CUDA
- Future Works

Self-Organizing Map

- Purpose
  - Make high dimensional data manageable
    - Neucleotide frequency count
    - 4^4 = 256 vector size
      - AAAA, AAAT, AAAC, AAAG, AATA, AATT, AATC, AATG...
  - visualize the characteristics in data
    - M x N x Vector size --(reduce)--> M x N
      - 150 x 350 x 256 reduces to a 150 x 350 image

SOM (cont.)

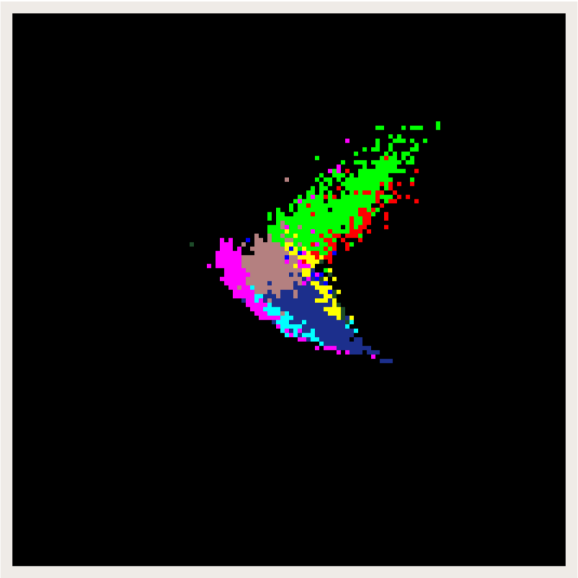
- Unsupervised learning
- Depending on the literature, neural network
- Produces 2D images from multi-dimensional data
  - While keeping the characteristics of the data intact

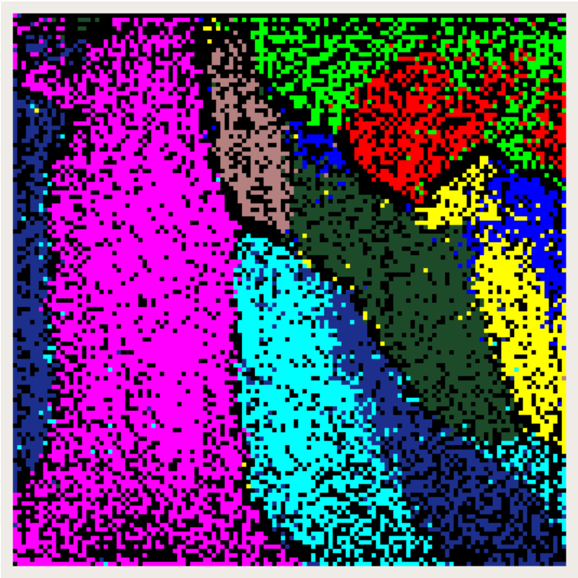
How a SOM is Used

- Characterize unknown data with known data
  - Similar to using a neural network
  - Color different data groups, run through SOM
  - Run unknown data, see where it lands in the SOM.

SOM (cont)

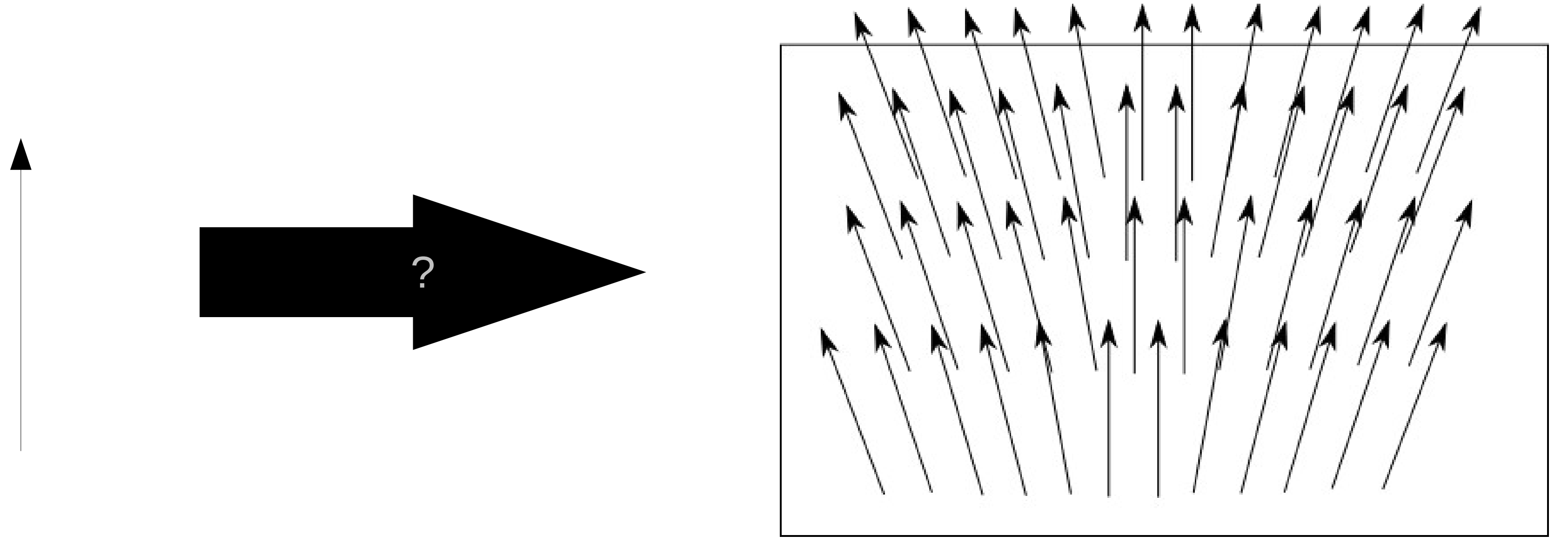
- 9 eukaryotic genomes
  - Vector size 256 (4^4) (only used 16 for images)
  - 40 iterations





Building a SOM, briefly

- Initialize the weight map with PCA
- For T iterations
  - For every vector in our data set
    - Find the “closest” weight vector
    - Adjust the weight vector plus its neightbors



Initial Weight Map

- Random or Principle Component Analysis?
  - Random may require more iterations
    - But should get you there anyways
  - PCA could create poor starting point
    - Vectors skewed via abnormally large portions of the data
    - Normalize count should take care of that
- I used PCA to create the initial weight vectors

Principal Component Analysis

- Is there another basis that reveals hidden structure in the data?
  - If we constrain this other basis to a linear combination of the original basis of the data, then yes
- We can use the covariance of the dataset
  - Then run PCA on the covariance

PCA (cont)

- Figure out the covariance and solve using SVD
  - Get new eigenvectors (we need the first two)
- Intialize Weight Map
  - $\vec{w}_{ij} = \vec{x}_{av} + \frac{5 * sigma_1}{I} * [\vec{b}_1 * (i - I/2) + \vec{b}_2 * (j - J/2)]$
  - Where b1 and b2 are eigenvectors of the first and second principal components, and sigma is the standard deviation.
  - X_{av} is the mean of the dataset
  - I,J are the weight map sizes

Distance Measure

- Given an M x N weight vector map and a data vector
  - How do you determine “close”
- As always, it depends on your data
  - I used $||(\vec{X}_k - \vec{W}_{ij})^2||$

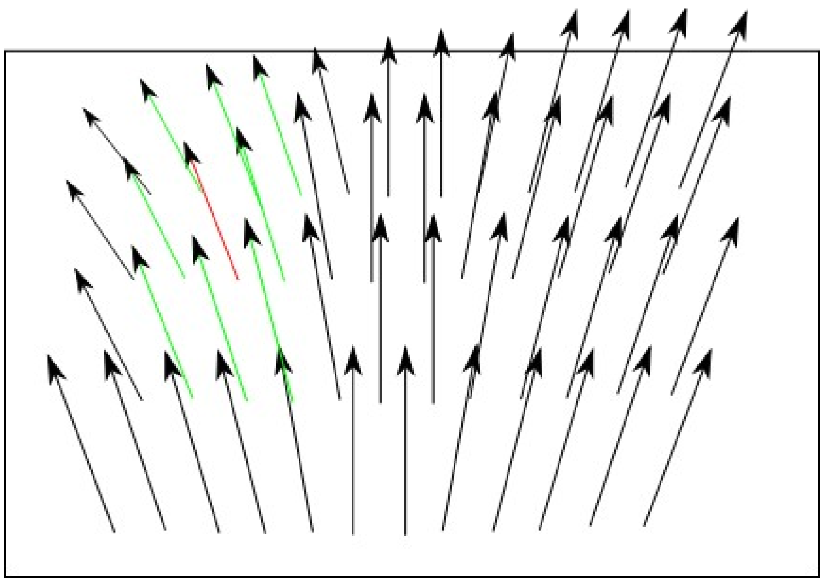
Update Weight Map

$$\overrightarrow{w_{ij}^{(new)}} = \overrightarrow{w_{ij}} + alpha(r) * \left(\frac{\sum \vec{x}_k}{N_{ij}} - \overrightarrow{w_{ij}} \right)$$

- Where alpha is a learning coefficient
  - After every iteration, alpha is updated
  - alpha(r) = max(0.01, alpha(1)*(1- r/T)
    - Where alpha(1) is the initial value of alpha, and T is the total number of iterations
- r is the iteration number

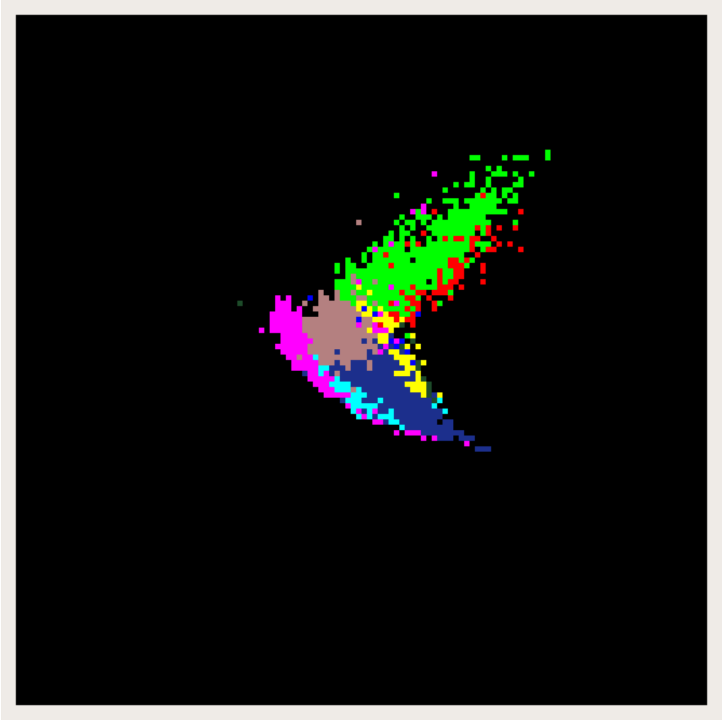
Update Weight Map (cont)

- Use a window
  - Use beta to control the size
  - After each iteration beta is updated
    - Beta(r) = max(0, Beta(1) – r)
    - Where Beta(1) is the initial value of beta



Other Useful Information

- Density



CUDA?

- PCA is host side (C++), SOM is CUDA
- Difficulties
  - Memory constraints in general
  - Memory constraints using SGEMM
    - Weight map width * weight map height * size of data
    - 64 x 64 x ~10,000 = ~3.6GB of RAM
  - Reduction required using SGEMV
  - Rolling my own matrix multiplier

Future works

- stuff that needs to be finished soon.
  - Finish writing a CUDA PCA
  - Do some speed measurements
- Big picture
  - Adapt it for detecting “fracture”
    - Canny Edge Detection

Questions?