CS 179 Lecture 15

Set 5 & machine learning

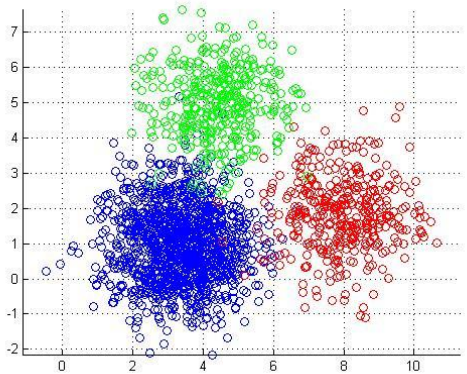
Set 5 goals

Practice overlapping computation with data movement on a streaming workload…

while building a machine learning system

Set 5 description

Cluster a stream of business review from Yelp.



Representing text as vectors

Yelp reviews are text. How can we quantify the similarity between two snippets of text?

“The doctor has horrible bedside manner”

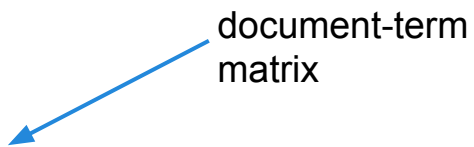
“The enchiladas were the best I’ve ever had!”

One solution is to represent the snippets as numerical vectors.

Bag of words approach

A very simple but very common approach. Count occurrences of each word

	cats	dogs	hats	love	eat
cats love hats	1	0	1	1	0
cats eat hats	1	0	1	0	1
cat eat cats	2	0	0	0	1



Loses all information on ordering of words.

Latent Semantic Analysis

Idea: Compute the singular value decomposition (SVD) of the document-term matrix. Singular values correspond to words that commonly appear together, which oftentimes correspond to our notion of a topic.

This is called latent semantic analysis.

Represent each document as coefficients of top-k singular vectors.

Clustering

Group data points based on some sort of similarity. Clustering is a very common unsupervised learning problem.

Many variants of clustering:

- hard/soft
- hierarchical
- centroid
- distribution (Gaussian mixture models)
- density

k-means clustering

Very popular centroid-based hard clustering algorithm.

```
def k_means(data, k):
    randomly initialize k “cluster centers”
    while not converged:
        for each data point:
            assign data point to closest cluster center
        for each cluster center:
            move cluster center to average of points in cluster
    return cluster centers
```

Stream clustering

Can we cluster if we can only see each data point once?

```
def sloppy_k_means(int k):
    randomly initialize k “cluster centers”
    for each data point:
        assign data point to closest cluster center
        update closest cluster center with respect to data point
    return cluster centers
```

This algorithm will give poor clustering.

Cheating at streaming algorithms

Idea: Use a streaming algorithm to create a smaller dataset (sketch) with properties similar to stream. Use expensive algorithm on sketch.

```
k_means(sloppy_k_means(stream, 20 * k), k)
```

Strategy described here, used by Apache Mahout. If length of stream is known (and finite), use `k*log(n)` clusters for sketch.

Parallelization

```
def sloppy_k_means(int k):
    randomly initialize k “cluster centers”
    for each data point:
        assign data point to closest cluster center
        update closest cluster center with respect to data point
    return cluster centers
```

What can we parallelize here?

Batching for parallelization

```
def sloppy_k_means(int k):
    randomly initialize k “cluster centers”
    for each batch of data points:
        par-for point in batch:
            assign data point to closest cluster center
        update cluster centers with respect to batch
    return cluster centers
```

Batching

Changing the computation slightly to process a batch of data at a time is a common theme in parallel algorithms.

Although it does change the computation slightly, batching still leads to some sort of local minima of the loss function.

When you already aren’t going to find an optimal solution, cutting corners isn’t that bad :)

Data transfer issues

Your program for set 5 will read LSA representations of reviews from stdin and will sloppy cluster them.

Your tasks:

- overlap data transfer and computation between host and device (hopefully saturate interface on haru)
- implement sloppy clustering algorithm
- analyze latency and throughput of system

Final comments on set

Set 5 should be out Tuesday evening.

Details are still getting worked out. Might involve using multiple GPUs. Will be relatively open-ended.

General machine learning on GPU

Already seen k-means clustering…

Many ~~machine learning~~ numerical algorithms rely on just a few common computational building blocks.

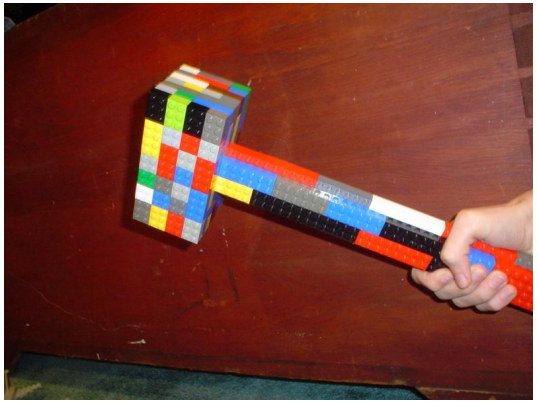
- element-wise operations on vectors/matrices/tensors
- reductions along axes of vectors/matrices/tensors
- matrix multiplication
- solving linear systems
- convolution (FFT)

Computational building blocks

These building blocks are why scientific scripting (MATLAB or Numpy) is so successful.

Often want to use the GPU by using a framework rather than writing your own CUDA code.

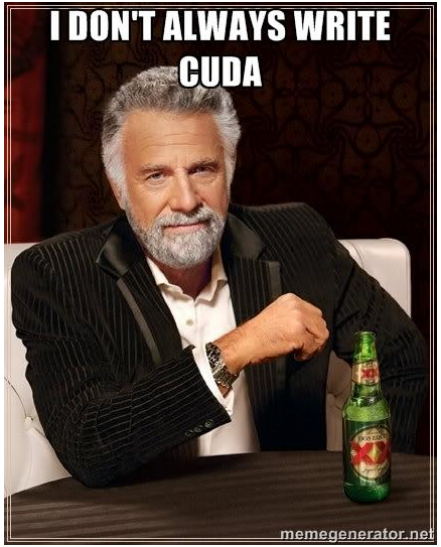
image from Todd Lehman



When to NOT write your own CUDA

Heuristic: If you could write it efficiently in “clean” MATLAB, you could likely accelerate it solely through using libraries either from NVIDIA (cuBLAS, cuSPARSE, cuFFT) or the community (Theano, Torch)

Better to write less CUDA and then call into a lot



When to write your own CUDA

Vectorized scripting languages must use a lot of memory when considering all combinations of input.

Example: What is the maximum distance between pairs of n points?

Most vectorized MATLAB implementations will store all n^2 distances, and then compute the maximum over these. Quadratic memory and time.

An implementation in a language with loops (that you actually want to use) takes linear memory and quadratic time.

What this week will cover

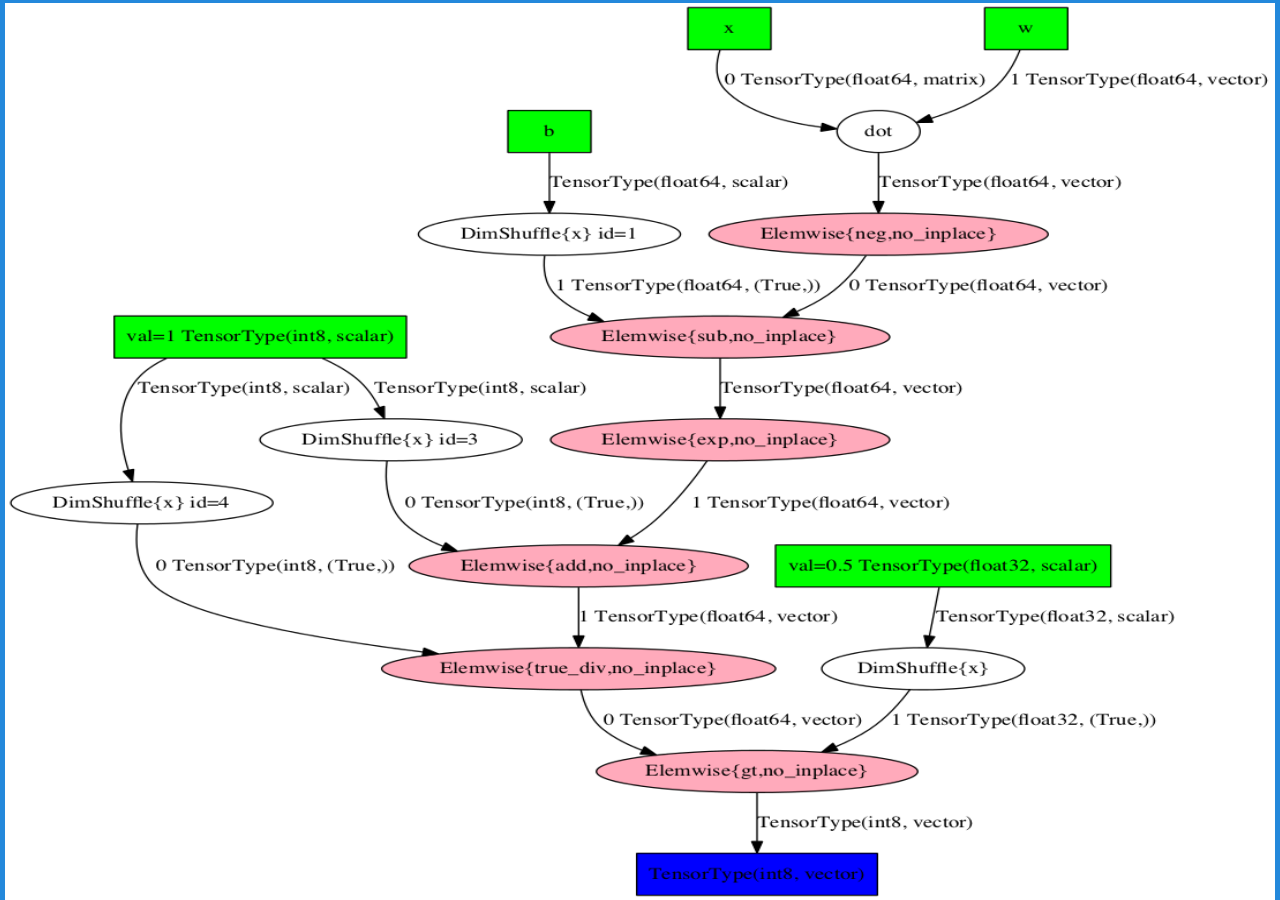
Today: Theano (Python library that makes GPU computing easy)

Rest of week: parallelizing machine learning yourself, how to write your own CUDA code

Theano

Python library designed primarily for neural nets, but useful for any sort of machine learning that involves training by gradient descent.

Key idea: Express computation as a directed graph through a Numpy-like interface.



A Theano computation graph (upper part computes exp(w*x - b))

Benefits of computational graph

Due to abstraction, can execute graph on any hardware for which all graph nodes are implemented. Multi-threaded C, GPU, etc. Can also optimize graph itself (and correct for numerical instability).

Can automatically compute gradient of any graph node output with respect to any other graph node (*automatic differentiation*)

Simple Theano example

```
import theano
import theano.tensor as T

input = T.matrix()
output = 1 / (1 + T.exp(-x))
logistic = theano.function([input], output)
logistic([[0.0, 1.0], [3.14, -1.0]])
```

Conclusion

Set 5 will hopefully be fun (esp since set 6 will be based on same code)

When working on an application, don’t write CUDA unless the complexity is necessary.

Theano is a useful Python library for doing scientific scripting on GPU