CS 179 Lecture 15

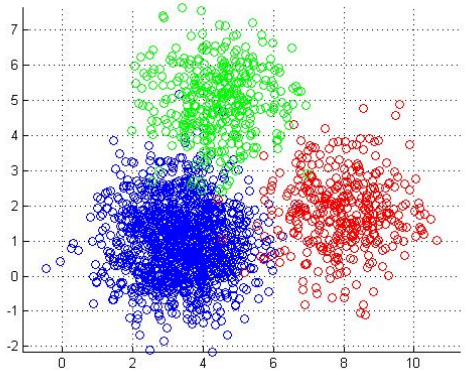
Set 5 & Machine Learning

Set 5 Goals

Practice overlapping computation with data movement on a streaming workload - while building a machine learning system.

2015 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

document-term matrix

Loses a ds.

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"
    foreach 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

2016 Set 5 Outline

- Logistic regression
- (Stochastic) gradient descent
- Parallelizing SGD for neural nets (with emphasis on Google’s distributed neural net implementation)

Binary Classification

Goal: Classify data into one of two categories.

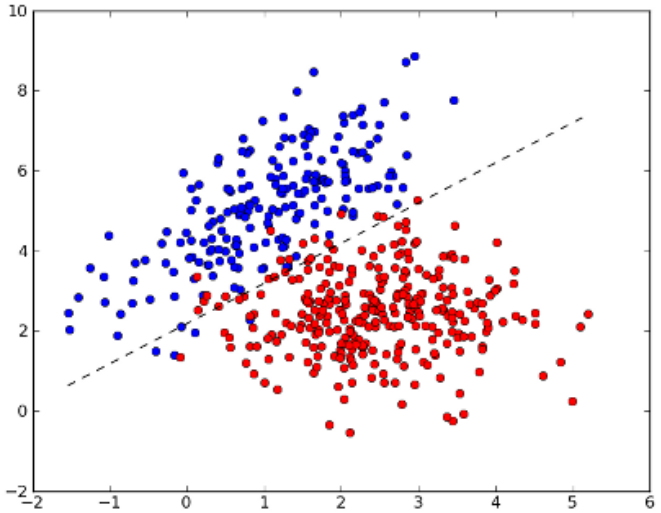
“Is this Yelp review of a restaurant or of a different type of business”?

Given:

- training set of (data, category) aka (X, y)
- test set of (X) for which we want to estimate y

Logistic Regression

There are many other binary classification algorithms (random forests, SVM, Bayesian methods), but we'll study logistic regression.



Scalar Logistic Regression

$$p = \frac{1}{1 + e^{-x^T w}}$$

p: probability of belonging to category 1
x: data point as n component vector
w: learned weight vector

Vectorized Logistic Regression

$$p = \frac{1}{1 + e^{-X^T w}}$$

Let matrix `X` be `n x m` with each column being a separate data point.

Now `p` is an `m` component vector of probabilities.

Learning

How can we find an optimal weight vector `w` from our training set?

In what sense can `w` be optimal?

Loss Functions

Weights can only be optimal with respect to some objective function. In general, we call this function the “loss” and we try to minimize it with respect to weights.

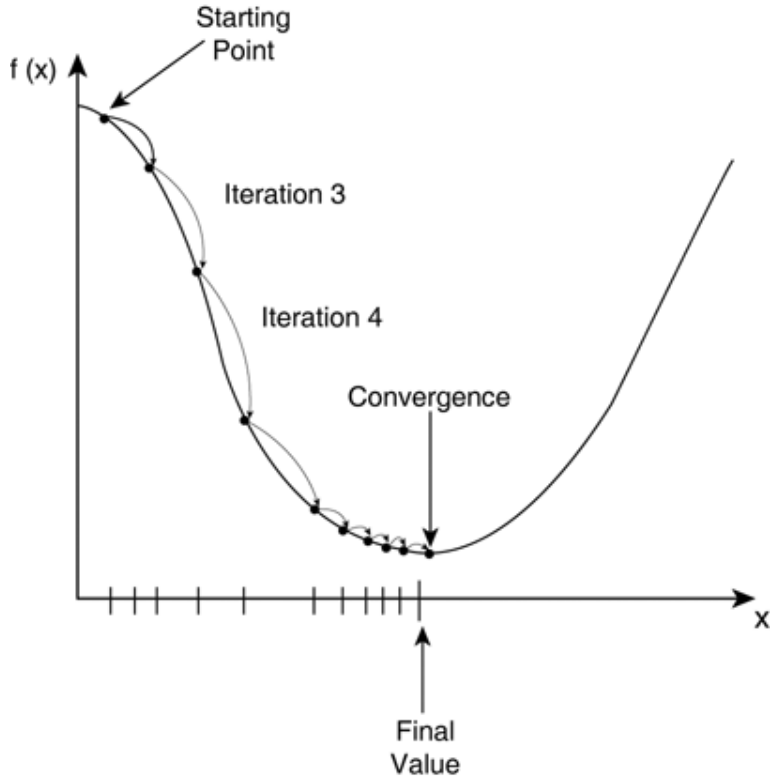
$$\mathcal{L} = \sum_i \log(1 + \exp(-y_i(x_i^T w)))$$

is the loss that gives logistic regression.

Gradient Descent

Compute the gradient of loss with respect to weights, and update weights in direction of negative gradient.

Repeat until you hit a local minima. Have to pick a step size.



Stochastic Gradient Descent (SGD)

The current formulation of gradient descent involves computing gradient over the entire dataset before stepping (called batch gradient set).

What if we pick a random data point, compute gradient for that point, and update the weights? Called stochastic gradient descent.

SGD Advantages

- easier to implement for large datasets
- works better for non-convex loss functions
- sometimes faster (you update the gradient much earlier and more incrementally)

Often use SGD on a “mini-batch” rather than just a single point at a time. Allows higher throughput and more parallelization.

Parallelizing SGD

2 (not mutually exclusive) routes:

- parallelize computation of a single gradient (model parallelism)
- compute multiple gradients at once (data parallelism)

Model Parallelism

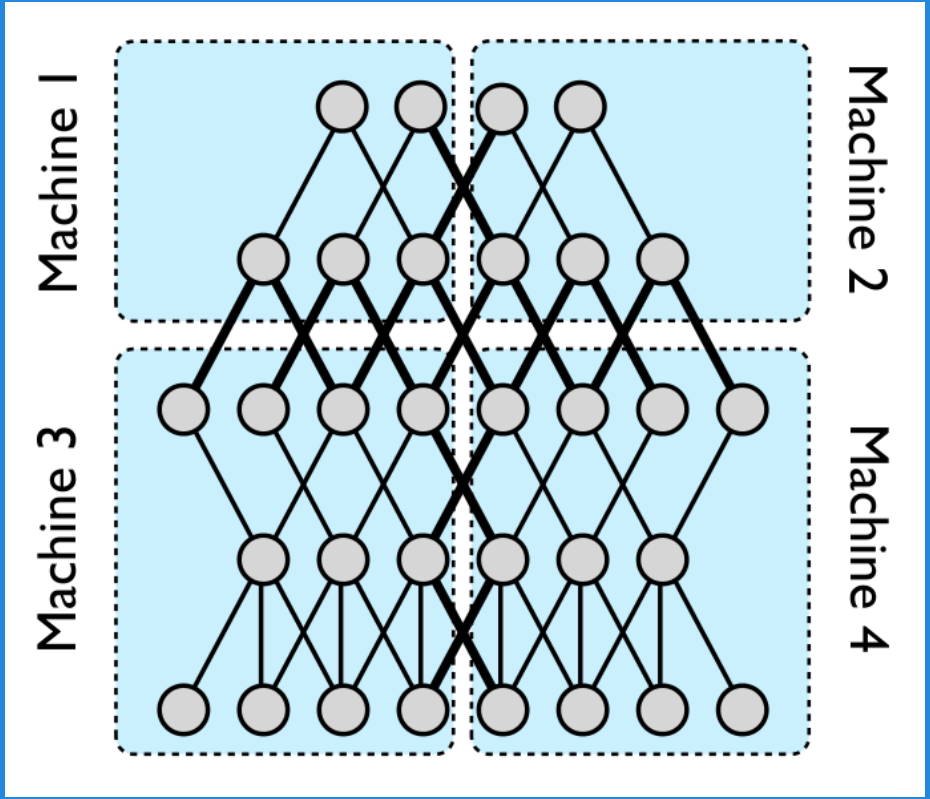
Model parallelism is “single model with parallel components”.

Can generally parallelize over the mini-batch.

Model “MATLAB Parallelism”

Many models (including logistic regression!) include matrix multiplication or convolution in gradient computation.

This is a good example of “MATLAB parallelism”, scriptable parallel computation built on top of a few kernels.



Model pipeline parallelism (in Google Brain neural nets)

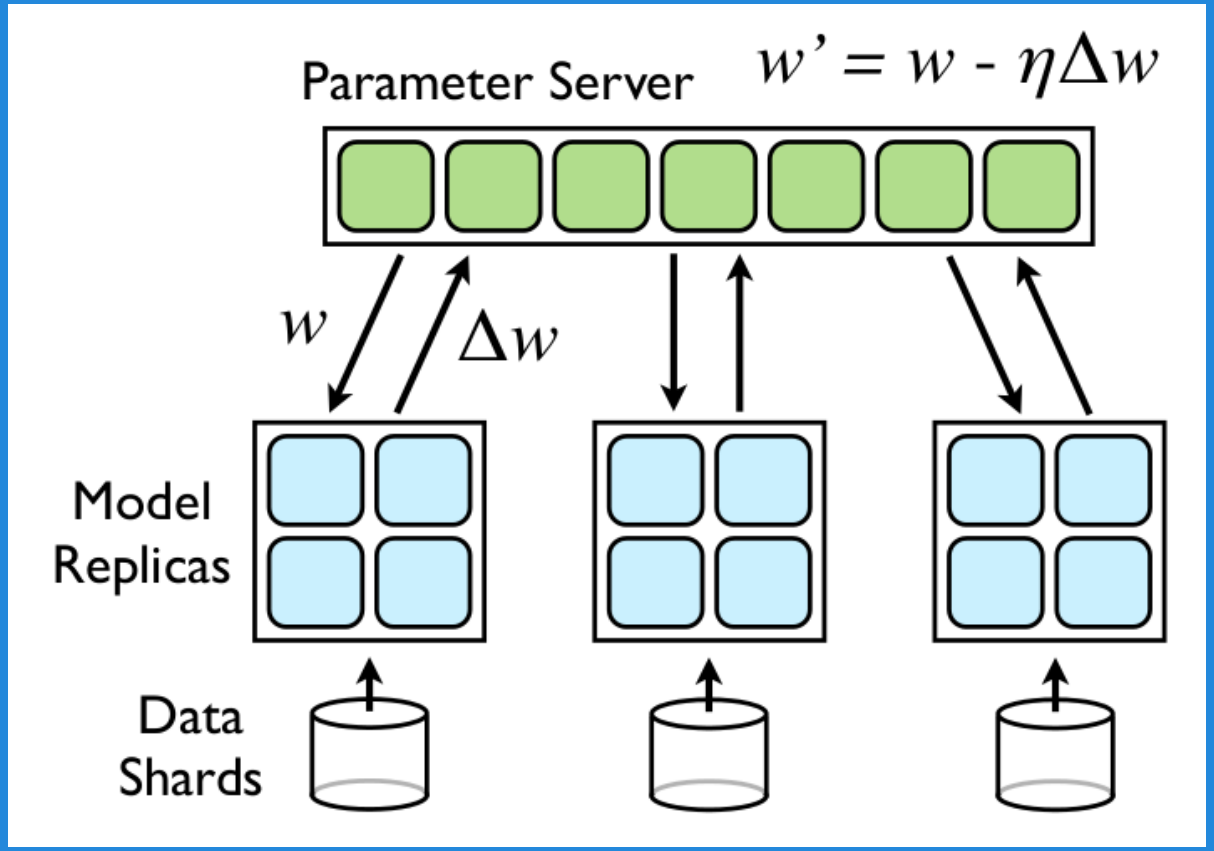
Data Parallelism

Run multiple copies of the model (that all share weights) on different data

Problem: SGD is very iterative. How do I synchronize updates to the weights?

Hogwild!

Some problems such as matrix completion have sparse gradients. A single output depends only on a single row and column of factorization.
Solution: Don’t worry about synchronization! Gradient updates unlikely to touch each other because of sparsity.



Downpour SGD from Google Brain

Downpour SGD

Store all weights on a “parameter server.”

Each model replica fetches updated weights from server every n_{fetch} steps and pushes gradient to server every n_{push} steps.

Not a lot of theoretical justification, but it works :)

Google Brain Parallelism Summary

Data parallelism - multiple model replicas communication with parameter server, using downpour SGD
Model pipeline parallelism - each replica consists of a group of machines computing parts of model
Model “MATLAB parallelism” - each part of each pipeline uses GPUs to process mini-batch in parallel

Check out the paper

Final Thoughts

“MATLAB parallelism” is by far the simplest parallelism and is what you want when you have a single GPU.

Other parallelization techniques needed for bigger systems.

Keep GPUs in mind when doing machine learning, can often get ~10x speed-ups.

General Machine Learning on GPU

Already seen k-means clustering…

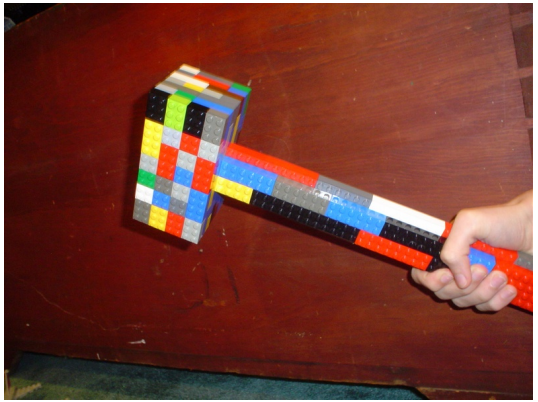
Many ~~machine learning~~ numerical algorithms rely on just a few common computational building blocks.

- 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.