CS 179 Lecture 16

Logistic Regression & Parallel SGD

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.

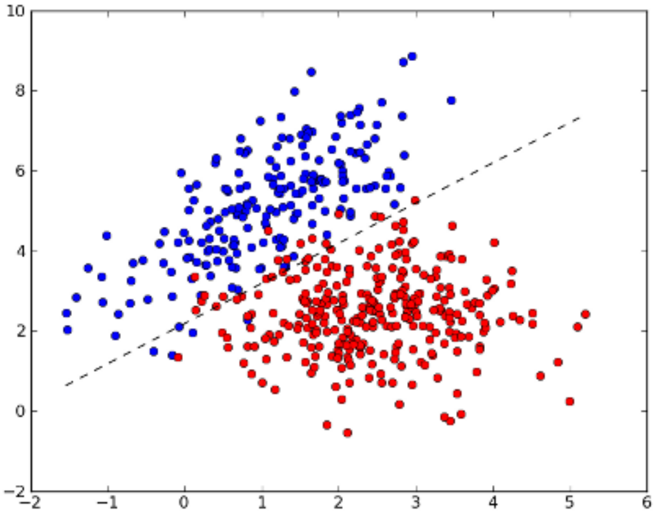
“Is this Yelp review of a restaurant or of a different type of business”?

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 seperate 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.

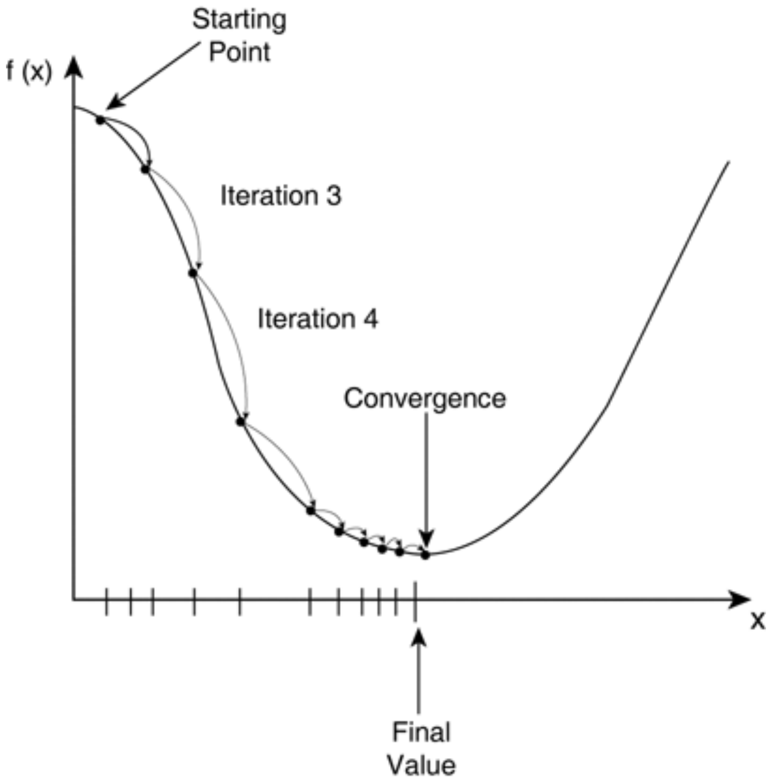
$$\mathcal{L} = \sum_i \log(1 + \exp(-y_i(x_i^T w)))$$

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

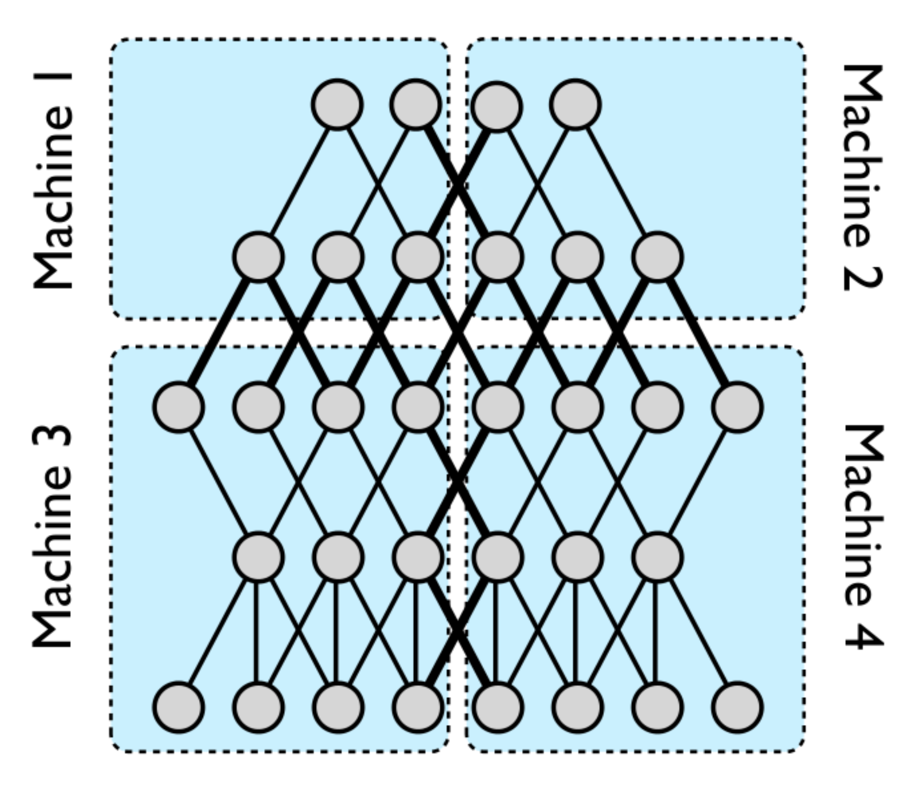
Model parallelism is “single model with parallel components”.

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

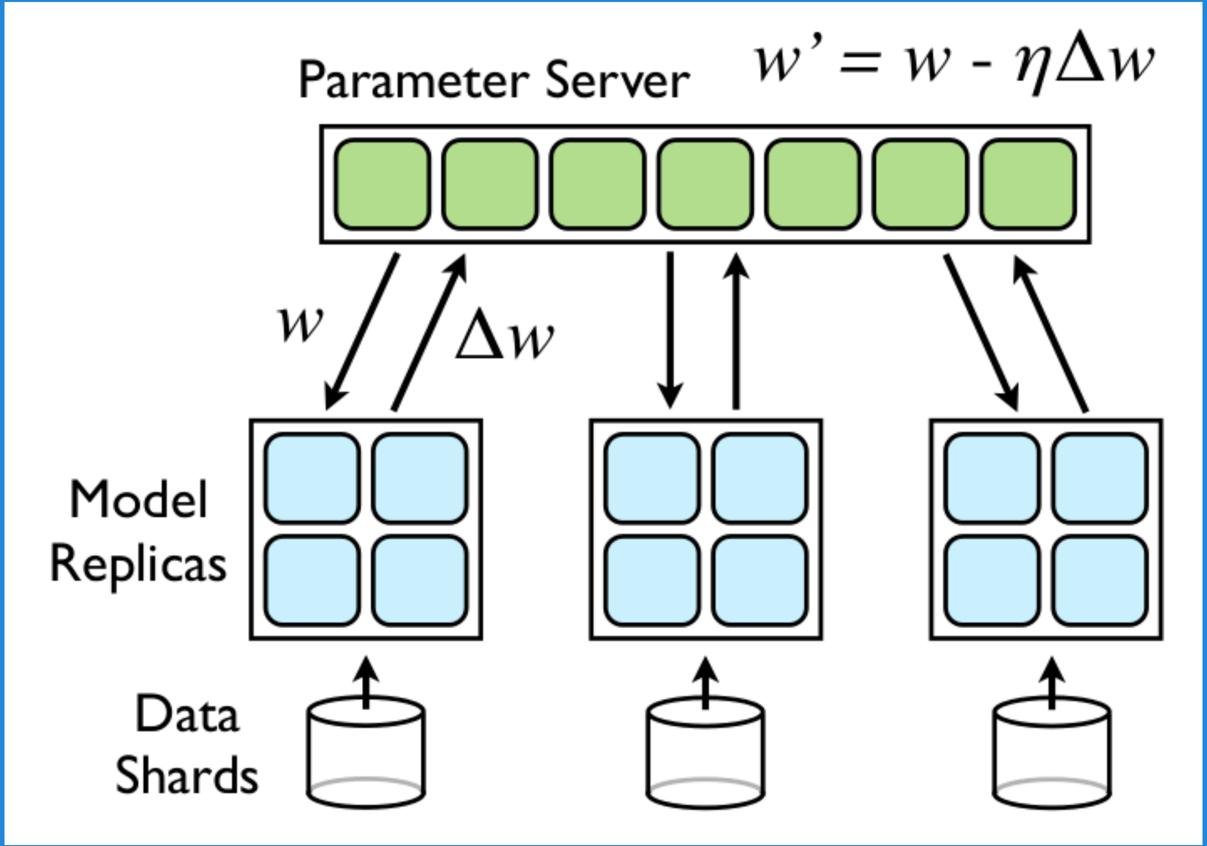
Run multiple copies of the model (that all share weights) on different data

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 $n_{\texttt{fetch}}$ steps and pushes gradient to server every $n_{\texttt{push}}$ 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.