Large-Scale Machine Learning

What do we need to do differently when $m \sim \mathcal{O}(10^8)$?

- Using a small data set chosen from the larger set, plot training curves and verify that the model has high variance.
  If it does not, a randomly-chosen subset will probably work

---

Stochastic Gradient Descent (as opposed to "Batch" gradient descent)

If m is large, fitting takes a long time.

$$\text{cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \rightarrow J_{TRAIN} = \frac{1}{m}\sum_{i=1}^{m} \text{cost}(\theta, (x^{(i)}, y^{(i)}))$$

1. randomly shuffle dataset
2. Repeat { % 1-10 passes common

   for $i = 1:m$

   $$\theta = \theta - \alpha\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)}$$

   } % take a step for each data point

The system converges to a random walk in the region of the global minimum.

Mini-Batch Gradient Descent

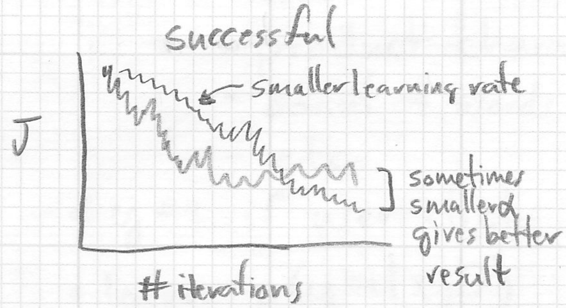
- Batch - use all m for each iteration
- Stochastic - use 1 for each iteration
- Mini-Batch - Use b examples in each loop ($b \in [2, 100]$) and use a new subset for each iteration.

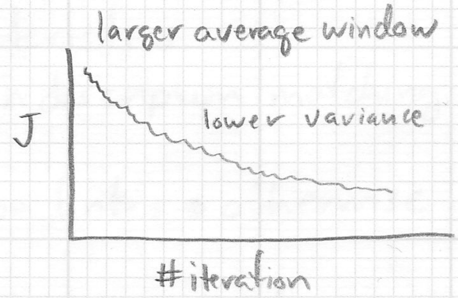
Tends to out-perform Stochastic gradient descent when the computation can be parallelized.

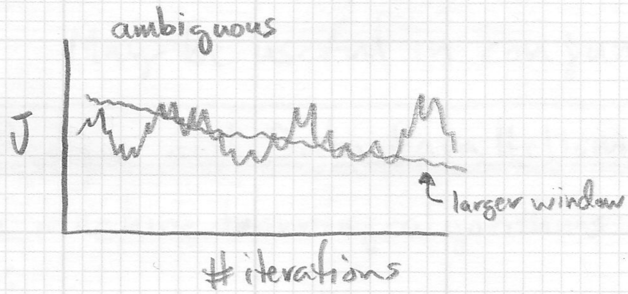
Stochastic Gradient Descent - Convergence

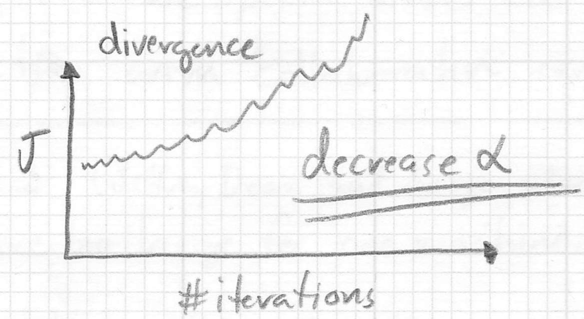
Computing J_{TRAIN} is expensive.

During learning, compute cost before update. Plot cost averaged over a number of steps.









Slowly decreasing α over time can improve convergence. (not common)

E.g. $\alpha = \frac{\text{const1}}{\text{iteration\#} + \text{const2}}$

Online Learning - continuous stream of data

Repeat forever {

- Get (x, y)
- Update θ using (x, y)

}

- don't save training data
- θ evolves over time (similar to stochastic gradient descent)

* can adapt to changing user preferences *

CTR → "Click Through Rate"

Example: Product Search (learning to search)

- User searches for a product
- Return top results
- make a feature vector for each returned result
  - product characteristics
  - count of words in description matching search terms
  - etc
- $y=1$ if user clicks on link, 0 otherwise
- learn $p(y=1|x;\theta)$
- Use θ to improve searches, or choose special offers etc.

May be useful in conjunction with collaborative learning.

Map Reduce and Data Parallelism

(Jeffrey Dean and Sanjay Ghemawat)

* Some machine learning algorithms are too large for one machine. *

Iterate:
- Divide data evenly among available machines/cores.
- On each machine:
  - iterate using its subset of data
  - send result for subset to master node
- master node combines result and updates parameters

* HADOOP *

Many learning algorithms can be expressed as a sum of functions over the training set. These are candidates for map-reduce parallelization.

Example: Neural Network

- do forward/back propagation on subsets of data
- combine results on master node