Map Reduce and Data Parallelism

Previously spoke about stochastic gradient descent and variations of the stochastic gradient descent algorithm, including those adaptations to online learning, but all of those algorithms could be run on one machine, or could be run on one computer.

And some machine learning problems are just **too big to run on one machine**, sometimes maybe we just don't ever **want to run all that data through a single computer**, no matter what algorithm we would use on that computer. We're going to see about different approach to large scale machine learning, called the **map reduce approach**.

Map-reduce

We want to do batch gradient descent

Batch gradient descent: $\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

We will assume that m = 400

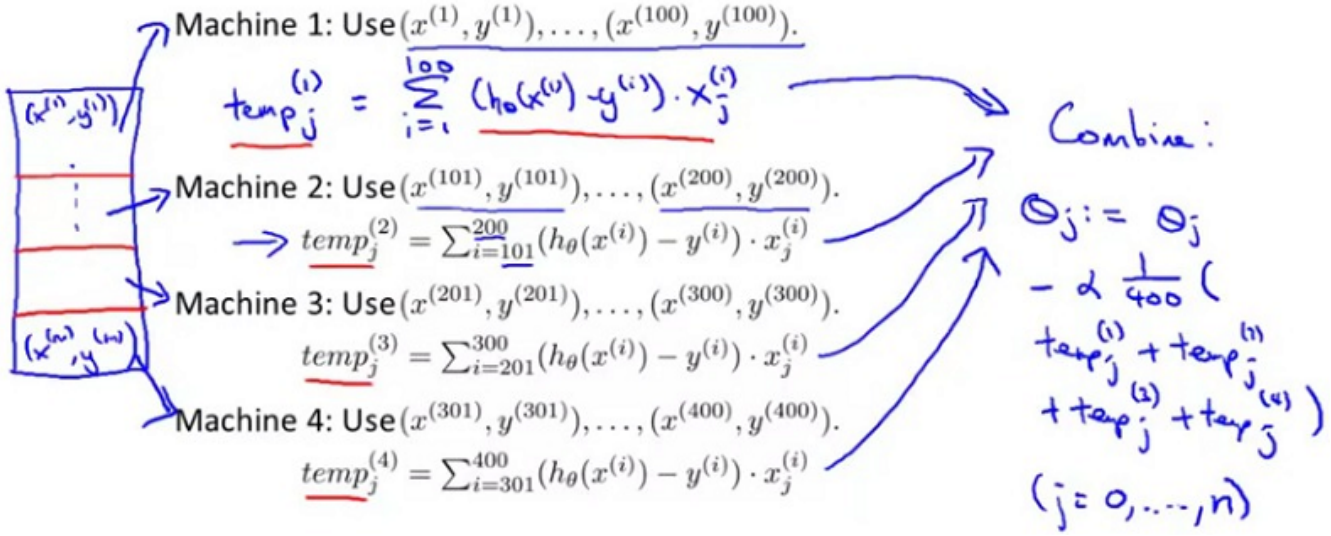
- Normally m would be more like 400,000,000
  - So, if m is large this step is really expensive

Let's say we have some training set (where m = 400):

- **In the MapReduce idea, we split the training set in to different subsets.**
- In this example we're going to assume that we have 4 computers
- So split training set into 4 pieces

Machine 1: use $(x^{(1)}, y^{(1)}), \ldots, (x^{(100)}, y^{(100)})$

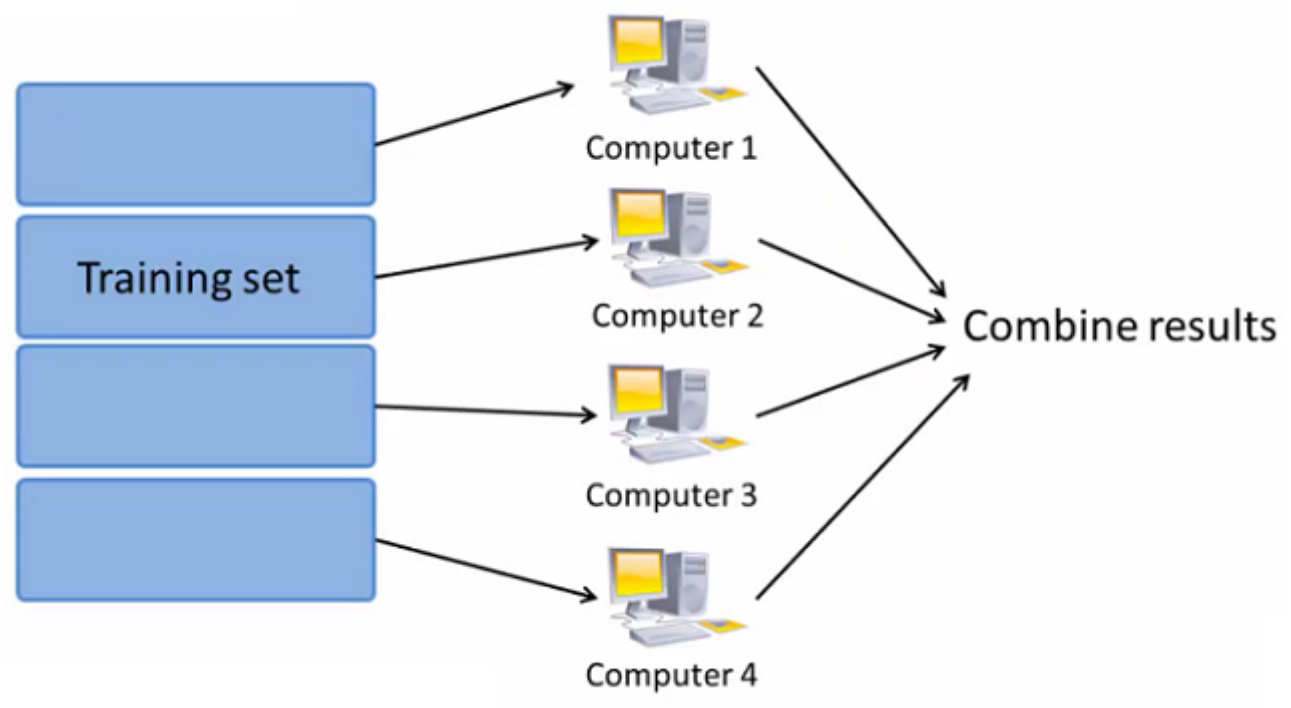
- Uses first quarter of training set
- Just does the summation for the first 100
- $temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$



So now we have these four temp values, and each machine does 1/4 of the work

- Once we've got our temp variables
- Send to to a centralized master server and put them back together

More generally map reduce uses the following scheme (e.g. where we split into 4)



The bulk of the work in gradient descent is the summation $\sum_{i=1}^{400}(\ldots)$

- Now, because each of the computers does a quarter of the work at the same time, we get a 4x speedup
- In practice, because of network latency, combining results, it's slightly less than 4x
  - **But nonetheless, this sort of map-reduce approach does offer us a way to process much larger datasets than is possible using a single computer.**

Map-reduce and summation over the training set

Can our learning algorithm be expressed as a summation over the training set?

- **Our learning algorithm is MapReduceable if it can be expressed as computing sums of functions over the training set. Linear regression and logistic regression are easily parallelizable.**

Let's look another example:

- E.g. for advanced optimization, with logistic regression, need:

$$J_{train}(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) - (1-y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

$$\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

  Need to calculate cost function - see we sum over training set
- So split training set into x machines, have x machines compute the sum of the value over $1/x^{th}$ of the data.
- These terms are also a sum over the training set
- **So with these results send temps to central server to deal with combining everything**

Multi-core machines

More broadly, by taking algorithms which compute sums we can scale them to very large datasets through parallelization

- Parallelization can come from:
  - Multiple machines
  - Multiple computers in a computer cluster
  - Multiple computers in the data center

It turns out that sometimes even if we have just a single computer, MapReduce can also be applicable.

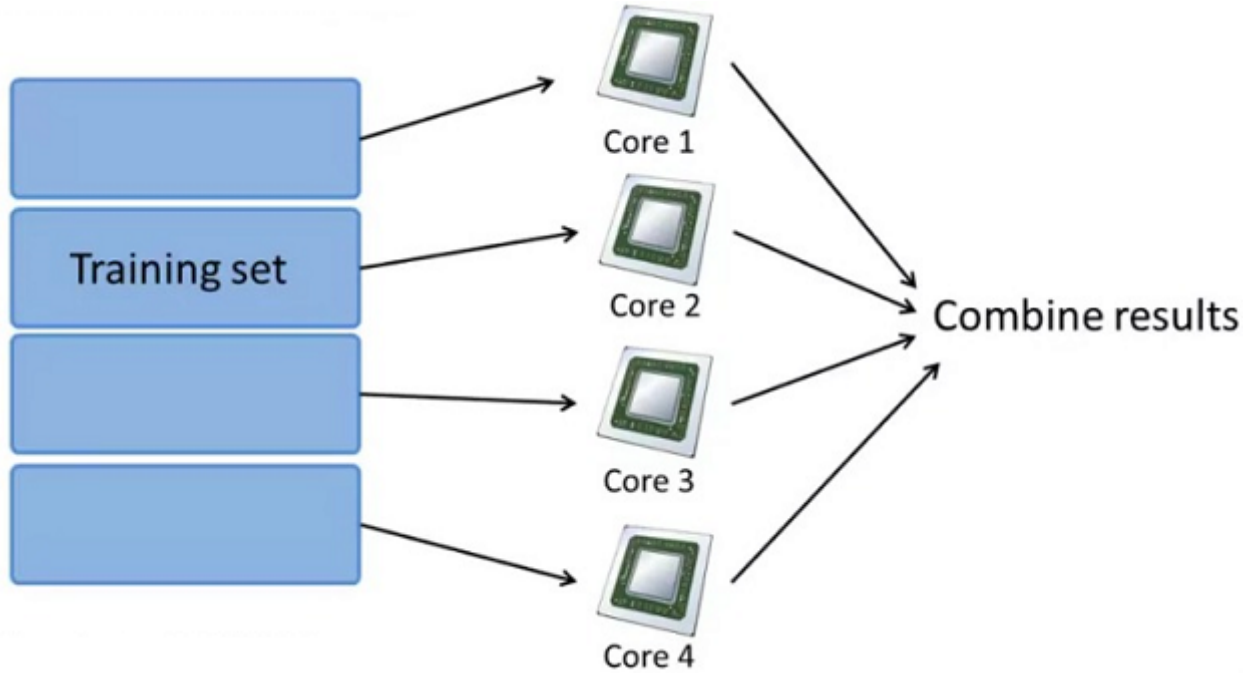
- In particular, on many single computers we can have multiple processing cores.
- We can have multiple CPUs, and within each CPU we can have multiple processing cores.

If we have a large training set, what we can do is:

- Say we have a computer with 4 computing cores, what we can do is, **even on a single computer we can split the training sets into pieces**
  - send the training set to **different cores within a single box**, like within a single desktop computer or a single server and **use MapReduce this way to divy up work load.**

The advantage of thinking about Map Reduce here is because we don't need to worry about network issues

- It's all internal to the same machine



Finally caveat/thought:

- Depending on implementation detail, certain numerical linear algebra libraries can automatically parallelize our calculations across multiple cores.
- So, if this is the case and we have a good vectorization implementation we can not worry about local Parallelization and the local libraries sort optimization out for us.

Video Question: Suppose you apply the map-reduce method to train a neural network on ten machines. In each iteration, what will each of the machines do?

- Compute either forward propagation or back propagation on 1/5 of the data.

> Compute forward propagation and back propagation on 1/10 of the data to compute the derivative with respect to that 1/10 of the data.

- Compute only forward propagation on 1/10 of the data. (The centralized machine then performs back propagation on all the data).
- Compute back propagation on 1/10 of the data (after the centralized machine has computed forward propagation on all of the data).