Large scale machine learning

Learning with large datasets

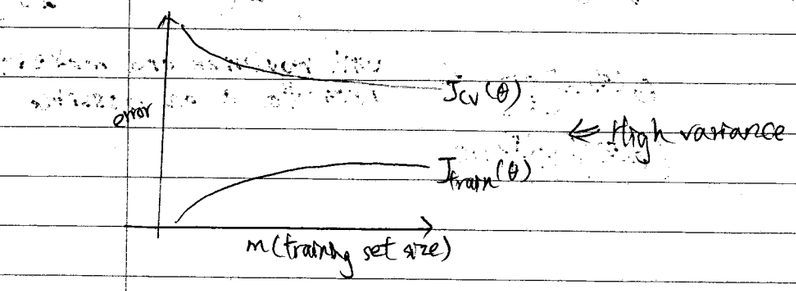
Let's say $m = 100,000,000$

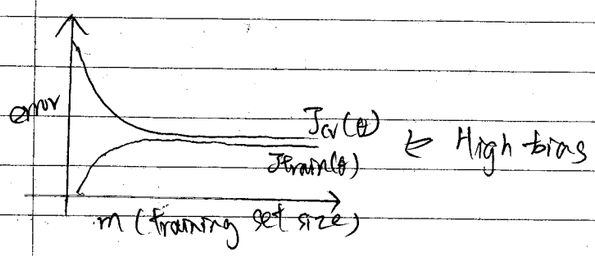
$\rightarrow \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

try take random $m = 1000$ and see will it works just well

Quiz Suppose you are facing a supervised learning problem and have a very large dataset ($m = 100,000,000$). How can you tell if using all of the data is likely to perform much better than using a small subset of the data (say $m = 1,000$)?

⇒ Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.





Large Scale Machine Learning

Stochastic Gradient Descent

Linear regression with gradient descent

$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$

$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$\frac{\partial}{\partial \theta_j} J_{train}(\theta)$

Repeat {

$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ ← expensive to compute

(for every $j = 0, \dots, n$)

}

$m = 300,000,000$ (very large)

Batch gradient descent

refer to: we look at all of the training examples at a time (look at all examples in every single iteration)

⇒ look through all training examples and minimize parameter to minimum every time

Stochastic gradient descent

$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$ → measures how well is hypothesis doing on single example $(x^{(i)}, y^{(i)})$

$J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$

1. Randomly shuffle dataset (m training set) (reorder)

2. Repeat {

Can repeat 1–10 times

But if m very, very large, might need to do only one

for $i = 1, \dots, m$ {

$\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$ → $\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$

(for $j = 0, \dots, n$)

}

}

⇒ Scan example 1 $(x^{(1)}, y^{(1)})$, modify parameter to fit better (just 1st)

⇒ Scan $(x^{(2)}, y^{(2)})$, modify parameter to fit it better (just 2nd)

⋮

⇒ Go through the entire training set

⇒ one by one go through each training example and minimize parameter each time. (Just to fit each single training example better)

⇒ Not converge in the same sense as Batch gradient descent does

⇒ ends up wandering around continuously in some region close to global minimum, but doesn't just get to global minimum and stay there

(So as parameters that end up pretty close to the global minimum, that will be a good hypothesis)

Large scale machine learning

Mini-batch gradient descent

Batch gradient descent: Use all m examples in each iteration
Stochastic gradient descent: Use 1 example in each iteration
Mini-batch gradient descent: Use b examples in each iteration
b = mini-batch size (2-100)

Get $b=10$ examples $(x^{(i)}, y^{(i)}), \dots, (x^{(i+9)}, y^{(i+9)})$

$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$

$i := i + 10$

Mini-batch gradient descent

Say $b=10$, $m=1000$

Repeat {

for $i = 1, 11, 21, 31, \dots, 991$ {

$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$

(for every $j = 0, \dots, n$)

}

}

If $m = 300,000,000$, you can starting making progress in modifying parameters after looking at just 10 examples rather than wait till you've scan through every single training example of 300 million of them.

⇒ Mini-batch gradient descent is likely to outperform Stochastic gradient descent only if you have a good vectorized implementation

Disadvantage ⇒ extra parameter b, mini-batch size which have to fiddle with, therefore take more time.

Large scale machine learning

Stochastic gradient descent convergence

Checking for convergence

Batch gradient descent:

Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.

$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Stochastic gradient descent: (Doesn't converge to global minimum)

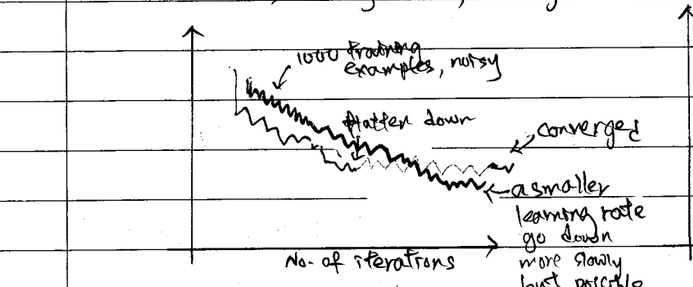
$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$

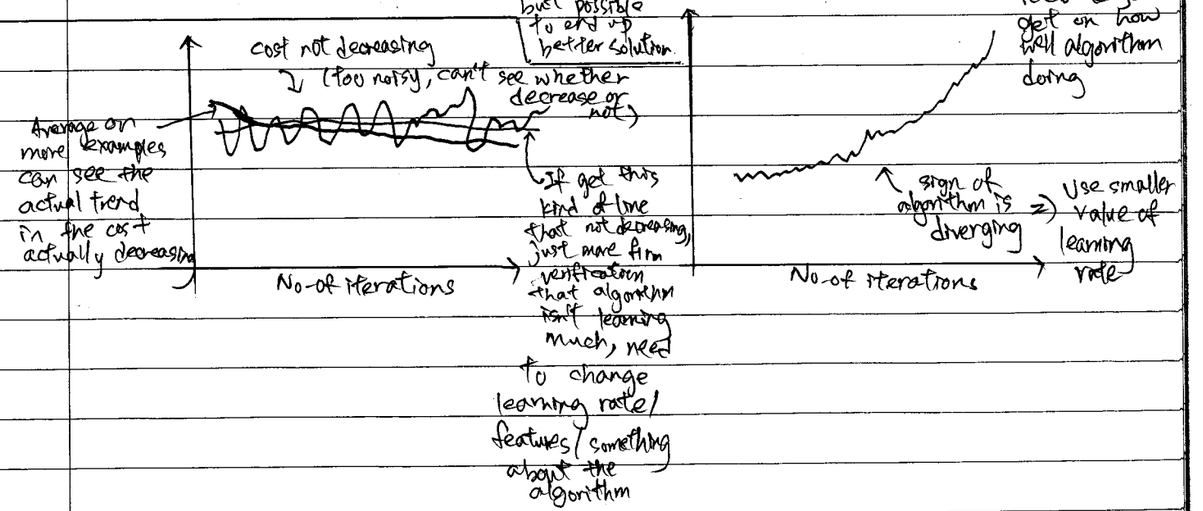
During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.

~~Checking for convergence~~

⇒ Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.

Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples





⇒ Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (To global minimum) E.g. $\alpha = \frac{const1}{iterationNum + const2}$

Large scale machine learning

Online learning

⇒ to model problems where we have continuous stream of data coming in

⇒ Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ($y=1$), sometimes not ($y=0$).

⇒ Features x capture properties of user, of origin/destination and asking price. We want to learn $p(y=1|x;\theta)$ to optimize price.

example: Keep staying up to website

⇒ Logistic Regression

Repeat forever {

Get (x,y) corresponding to user ← origin and destination specified by user, price offer to user → I/O user choose the shipping service or not

Update θ using (x,y):

$$\theta_j := \theta_j - \alpha(h_\theta(x)-y)\cdot x_j \quad (j=0,\cdots,n)$$

}

1. Look at one example at a time
2. Learn from that example
3. Discard it
4. That's why we not using fixed sequences like $(x^{(1)},y^{(1)})$

⇒ Can adapt to change user's preference and keep track of what your changing population of users may be willing to pay for new price.

Other online learning example:

Product search (learning to search)

User searches for "Android phone 1080p camera"

Have 100 phones in store. Will return 10 results.

x = features of phone, how many words in user query match name of phone, how many words in query match description of phone, etc. 10 (x,y)

$y=1$ if user clicks on link. $y=0$ otherwise. [Predicted CTR]

Learn $p(y=1|x;\theta)$ ← (problem of learning predicted click-through rate)

Use to show user the 10 phones they're most likely to click on

⇒ Continuously show user, 10 best guesses for phones users might like, each time user come, would get 10 examples (x,y) and use online learning algorithm to update the parameters using 10 steps of gradient descent on these 10 examples, and throw old data away.

Other examples: Choosing special offers to show user; customized selection of news articles; product recommendation; .....

Large scale machine learning

Map-reduce and data parallelism

Map-reduce

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

Machine 1: Use $(x^{(1)}, y^{(1)}), \dots, (x^{(100)}, y^{(100)})$.

$temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Machine 2: Use $(x^{(101)}, y^{(101)}), \dots, (x^{(200)}, y^{(200)})$

$temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Machine 3: Use $(x^{(201)}, y^{(201)}), \dots, (x^{(300)}, y^{(300)})$.

$temp_j^{(3)} = \sum_{i=201}^{300} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Machine 4: Use $(x^{(301)}, y^{(301)}), \dots, (x^{(400)}, y^{(400)})$

$temp_j^{(4)} = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

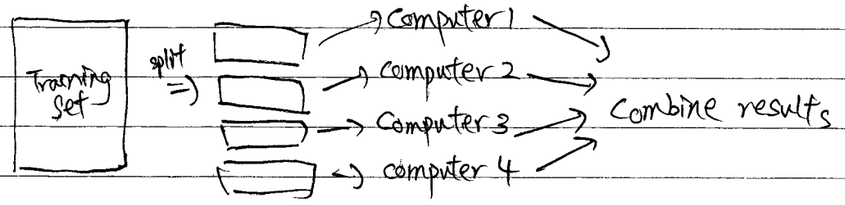
Same (At master server)

Combine:

$\theta_j := \theta_j - \alpha \frac{1}{400} \left[temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)} \right]$

$(j = 0, \dots, n)$

Map-reduce



Map-reduce and summation over the training set

Many learning algorithms can be expressed as computing sums of functions over the training set.

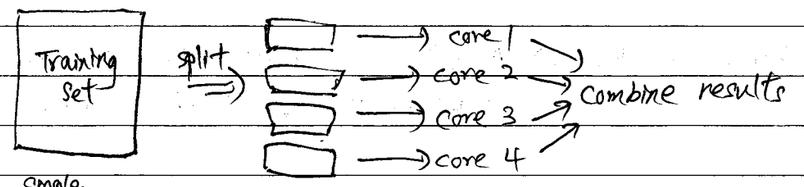
E.g. for advanced optimization, with logistic regression, need:

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

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

After each machine compute ↑, send to centralized server, which then add up the partial sums

Multi-core machines



(Single computer with 4 cores)

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

⇒ 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.