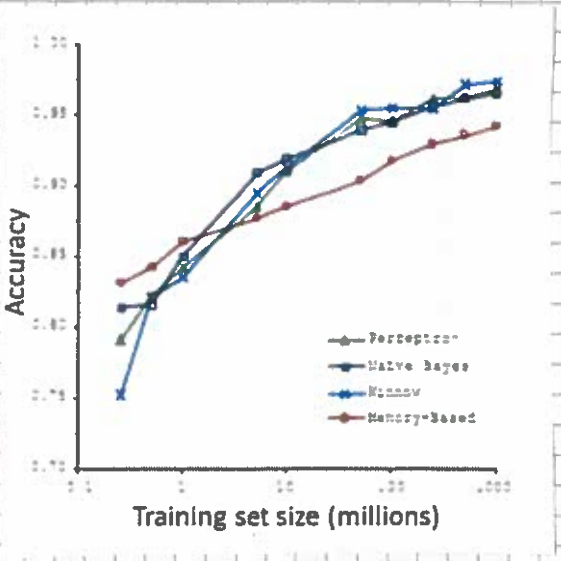
17 Large scale machine learning

Machine learning and data

Classify between confusable words.
E.g. {to, two, too}, {then, than}

For breakfast I ate "two" eggs.



"It's not who has the best algorithm that wins. It's who has the most data."

take low bias algorithm and train with lots of data.

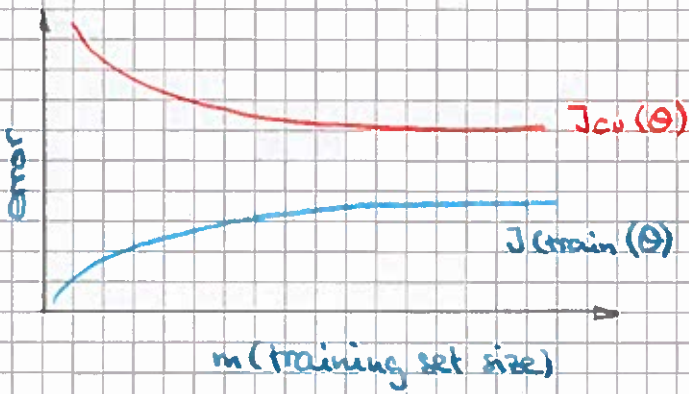
Learning with large data sets:

$m = 100'000'000$

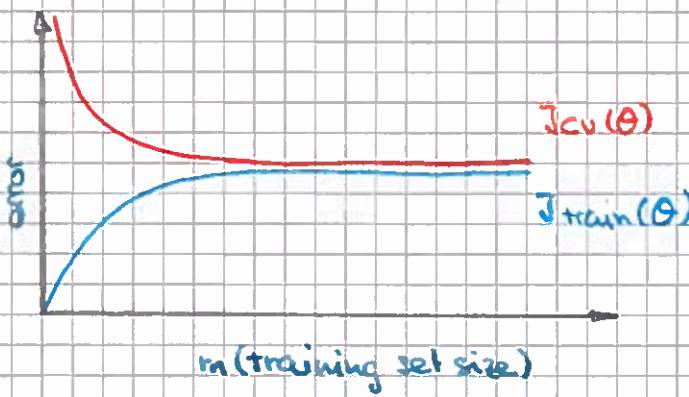
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

⇒ computationally expensive with $m = 100'000'000$

to use $m = 1000$ ⇒ sanity check



⇐ according to chapter 10 "high variance problem" can be solved with more data.



⇐ high bias no need to get more data or spend much time to optimize algorithm for $m = 100'000'000$. the $m = 1000$ would be sufficient.

add extra features or hidden units

Linear regression with gradient descent:

$$h_\Theta(x) = \sum_{j=0}^{n} \Theta_j x_j$$

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

Repeat {

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

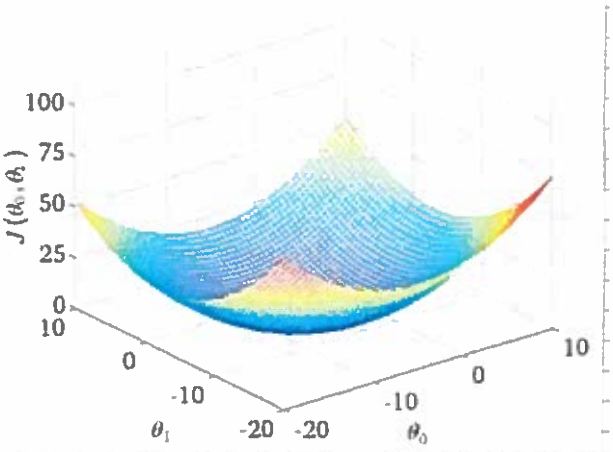
for every $j = 0, \dots, n$

}

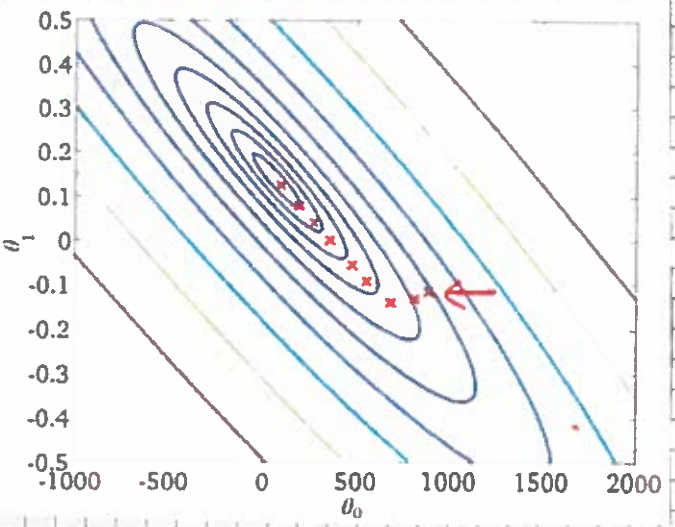
Summing over $m = 300'000'000 \Rightarrow$ data is computationally expensive

$\Rightarrow$ Batch gradient descent

Stochastic Gradient descent can also be used: Logistic regression, NN etc. (Algorithms based on GD)



$J(\Theta)$ cost function



- Batch gradient descent:
  Stochastic gradient descent:

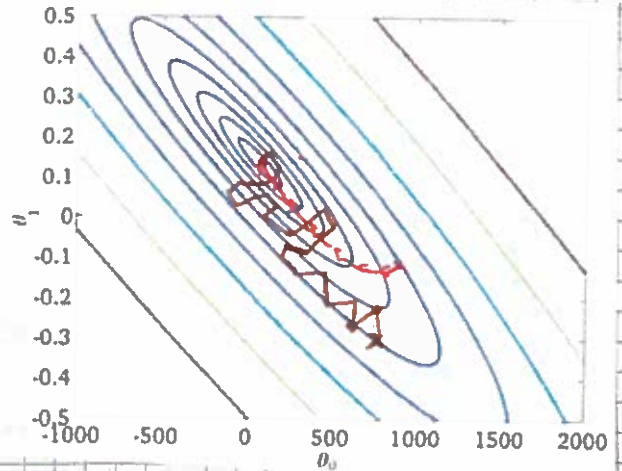
$$Cost(\Theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\Theta(x^{(i)}) - y^{(i)})^2$$

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

1) Randomly shuffle data set training examples
2) Repeat { → Repeat $1 - 10 \times$ (depending on training set size)
   for $i = 1, \dots, m$ {
   $\Theta_j := \Theta_j - \alpha (h_\Theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ $\Rightarrow$ *
   for $j = 0, \dots, n$
   }
   }

$$\rightarrow \frac{\partial}{\partial \Theta_j} cost(\Theta, (x^{(i)}, y^{(i)}))$$

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots$



$\Rightarrow$ in general converges to global minimum but not always.

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 => typically b = 10 but can be betw. 2 - 100

Mini batch gradient descent:

Say b = 10 m = 1000

Repeat {

For i = 1, 11, 21, ..., 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 $j = 0, \ldots, n$)

}

}

mini batch vs. stochastic implementation

- mini batch can only be better if you have a good vectorized implementation where you can parallelize steps. (partially)

Checking for convergence:

Batch gradient descent:

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

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \quad \Rightarrow m = 300{,}000{,}000$$

Stochastic gradient descent:

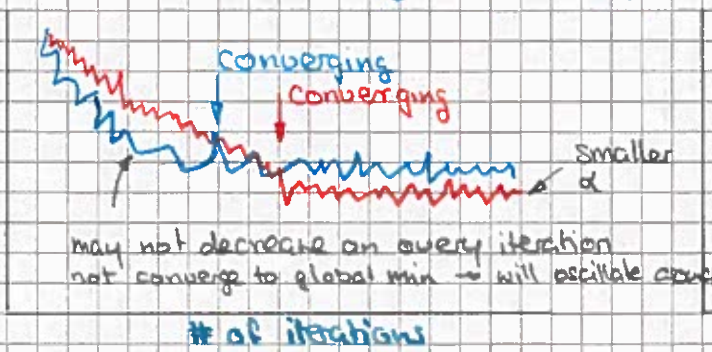
$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2 \quad \Rightarrow (x^{(i)}, y^{(i)}), (x^{(i+1)}, y^{(i+1)})$$

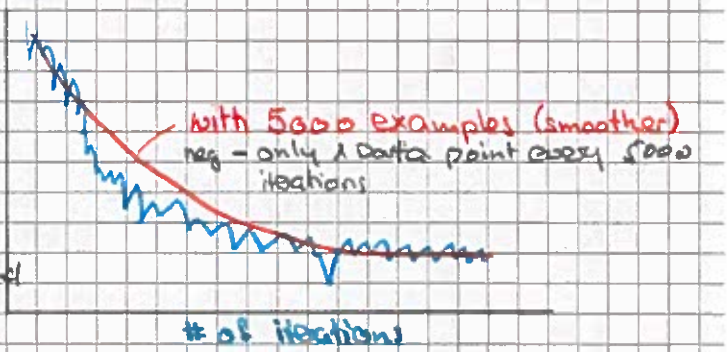
During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.

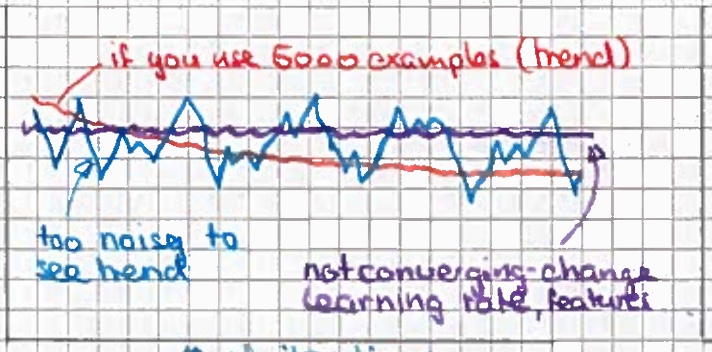
Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by the algorithm.

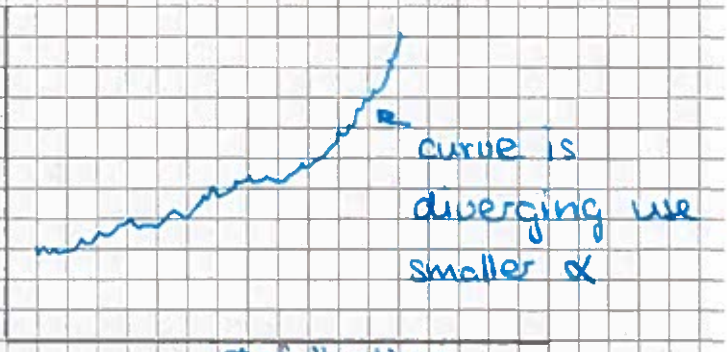
Checking for convergence:

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









Increase the number of examples to 5000 to see the trend?

Learning rate is typically held const. Can be slowly decreased to converge. (e.g. $\alpha = \frac{const\,1}{\#\,iterations + const\,2}$, $\alpha \to 0$)

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

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

Logistic regression or NN

Repeat forever {

Get (x,y) corresponding to user

Update θ using (x,y): ~~$(x^{(i)}, y^{(i)})$~~ because data is discarded.

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

}

$\Rightarrow$ Can adapt to changing user preferences.

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. $\Rightarrow$ gives us 10 (x,y) pairs

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

$y=1$ if user clicks on link. $y=0$ otherwise

Learn $p(y=1|x;\theta)$ $\Rightarrow$ predicted click through rate (CTR)

Use to show user the 10 phones they are most likely to click on.

Other examples: Choosing special offers to show users; customized selections of news articles; product recommendations...

Map reduce:

$m = 400$

realistic value: $m = 400'000'000$

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

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 $(x^{(201)}, y^{(201)}), \dots, (x^{(300)}, y^{(300)})$

$temp_j^{(3)} = \sum_{i=201}^{m=300}$

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

$temp_j^{(4)} = \sum_{i=301}^{m=400}$

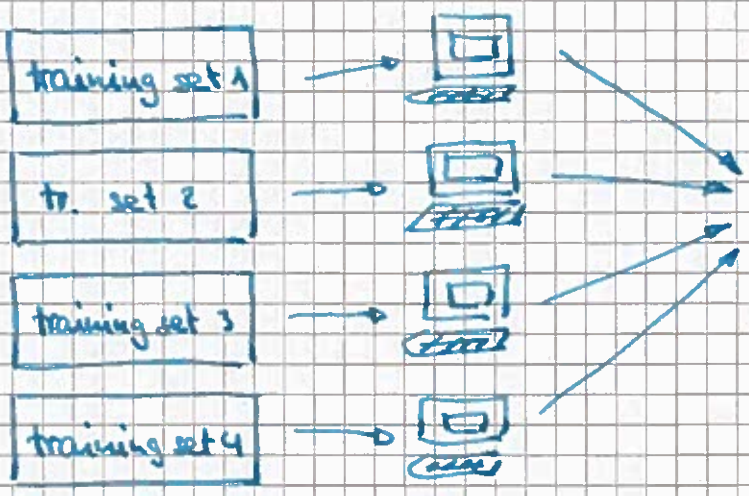
Combine: (On Master Server)

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

equivalent to batch gradient descent

$j = 0, \dots, n$

Map reduce:



Reduces the time by 4 times

(network latency, combine time ...)

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:

cost function

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

Partial derivative term

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

Sum of learning set → algorithm might be used for Map reduce

- LbFgs
- cost function

Multi core Machines: