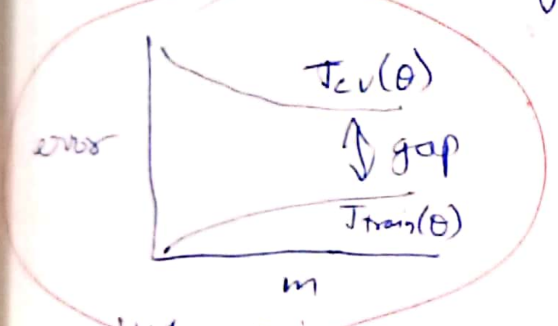
Week 10

Gradient Descent with large datasets

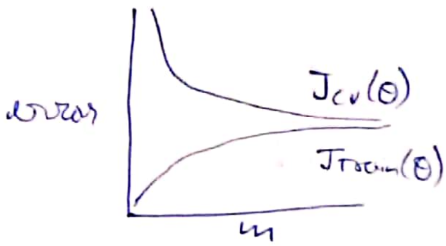
- Large datasets can improve accuracy

Lets say we have $m = 100$ million training examples. Inorder to check if more data can help, we need to check if it has high variance.

We do this by first taking a smaller dataset $m = 1000$, ploting the learning curve, if its high variance - only then will more data (100 million) result in better accuracy



High variance



High bias

Stochastic Gradient Descent

Batch gradient descent is very computationally expensive, since if we have $M = 300$ million training examples, it'll take 300 million iterations to get θ values

Linear Regression with gradient descent (batch)

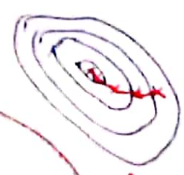
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Repeat {

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

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

}



Happens 300 million times

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 dataset
2. Repeat {

This is nothing but $\frac{\partial}{\partial\theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$

for $i := 1, \ldots, m$ {

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

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

}

}

Here for every iteration, it tries to fit one training example better

It doesn't need to reduce every step, it can even increase. But after a few iterations it reaches an area that is close the global minima

∴ We can repeat 1 ~~to~~ times

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 say $b = 10$, $m = 1000$.

Repeat {

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

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

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

}

}

- If $M = 300$ million, we can start making progress after only looking at $b=10$ examples rather than all 300 million for every iteration
- Mini-batch gradient descent is likely to outperform stochastic gradient descent only if you have a good vectorised implementation. So you can use use numerical algebra libraries and parallelize your gradient computations over b examples. You can use $B = 10$

Stochastic Gradient Descent Convergence

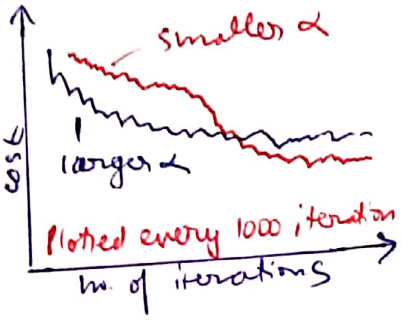
Checking for convergence:

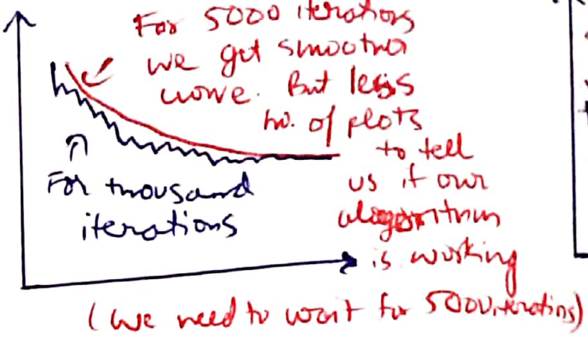
Before in batch gradient descent we would check if cost is decreasing every iteration. We can't do this for stochastic since a) It may not always decrease b) For large no. of examples (m), it can slow down the process.

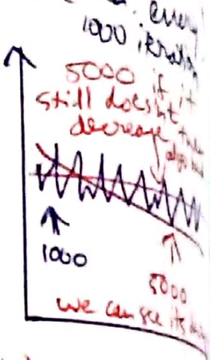
$\therefore$ Every 1000 iterations (lets say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm

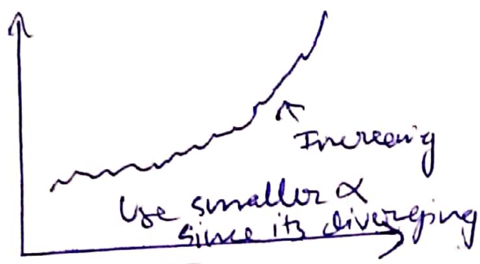
Different cases:

Plotting cost$(\theta, (x^{(i)}, y^{(i)}))$ over no. of iterations averaged over different no. of ~~plot~~ plots (ex: every 1000 iterations)









Before we found out that stochastic gradient descent doesn't converge to the global minima but just oscillated near it.

However one way to avoid this is by slowly decreasing $\alpha = \dfrac{\text{const1}}{\text{iterationNumber} + \text{const2}}$ and because of the smaller steps, it oscillates closer to the minima. However people don't use this since configuring const1 & const2 is extra work and the estimated minima given by Stochastic gradient descent is good enough

Online learning

If we have a stream of data coming we use this algorithm.

Repeat forever {

Get (x, y) corresponding to user

Update θ using (x, y)

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

}

It can adapt to changing user performance

Ex:

~~other~~

Product search

→ User searches for "Android phone 1080p camera"

→ we 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

→ $y = 1$ if user clicks on link, $y = 0$ otherwise

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

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

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

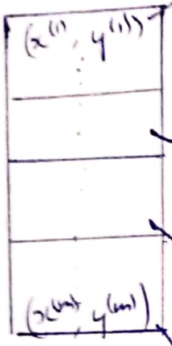
Map reduce & data parallelism

If we want to run the algo on multiple computers

Map reduce

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

Here we are taking $m = 400$ as an example but in reality this can be big like $m = 400$ million



Divide data into 4 parts

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

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

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

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

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

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

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

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

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

Centralised server

Combine

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

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

$\therefore$ You get an almost 4x speed (ignoring network latency, etc).

- You can use map reduce where summation occurs (summation is the computationally intensive)
- You can use it in a multi-core computer (Some numerical algebra libraries use this)