Week 10

- - @ learning with large datasets — — — —

#. large scale machine learning → Algo but viewing with big data sets.

• Dealing with massive datasets.

Best ways to get a high performance machine learning →
Take low bias learning algo, and train on lots of data.

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

It has computational problems.

learning with large datasets.

$m = 100,000,000$

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

A good sanity check → Randomly pick 1000 eg. of 100,000,000 and train our algo on 1000 eg. might do just as well.

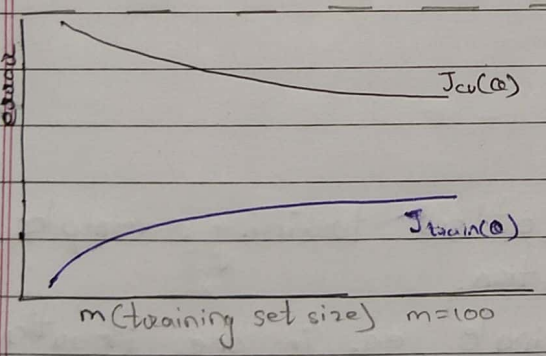
Q) Suppose you are facing a supervised learning problem and have a very large dataset (m = 100,000,000). How can you tell if using all the data is likely to perform much better than using a small subset of data (say m = 1,000)?

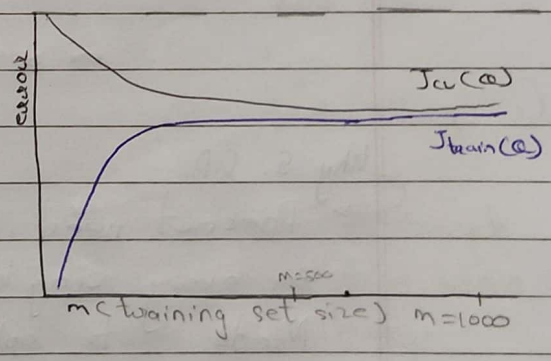
→ • There is no need to verify this; using large dataset always gives much better performance.

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

• Plot a learning curve ($J_{train}(\theta)$ and $J_{cv}(\theta)$), plotted as a function of m) for some range of values of m (say up to m = 1,000) and verify that the algorithm has bias when m is small.

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





High variance learning algo.
↓
adding extra terms
↓
Improve performance.

High bias learning algo
↓
will not increase performance.
↓
Stop

To go → Add extra features / hidden layers
Situation on left

And use more examples.

~~(Batch Gradient Descent)~~

--- (a) Stochastic Gradient Descent - - -

modification to basic gradient descent. (batch gradient descent)

applies to → logistic regn, neural network, ~~and~~ linear regn (training gradient descent)

focus of video

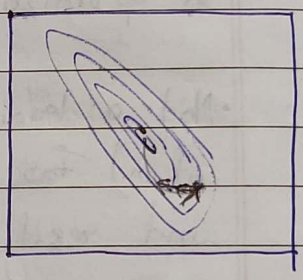
$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

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

Repeat {

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

}



m = 300,000,000

sum of 300000000 and single baby step,

for 1 iteration

epoch.

Why S. G.D.

→ Doesnot need to look at all training examples in every single iteration.

Needs to look at single eg. in one iteration.

Batch Gradient Descent

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

Repeat {

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

$$\frac{\partial J(\theta)}{\partial \theta_j}$$

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

}

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. (Good practice)
2. Repeat {

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

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

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

}

}

$$\frac{\partial\, cost(\theta, (x^{(i)}, y^{(i)}))}{\partial \theta_j}$$

i) Looking at only this first example $(x^{(i)}, y^{(i)})$, it's gonna take like a basically little gradient step. wrt cost of just first e.g.
To fit 1st e.g. a little bit better update θ's.

ii) $(x^{(2)}, y^{(2)})$

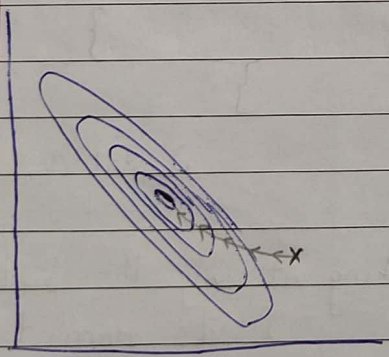
iii) $(x^{(3)}, y^{(3)})$

Repeat ⇒ takes multiple passes over entire training set

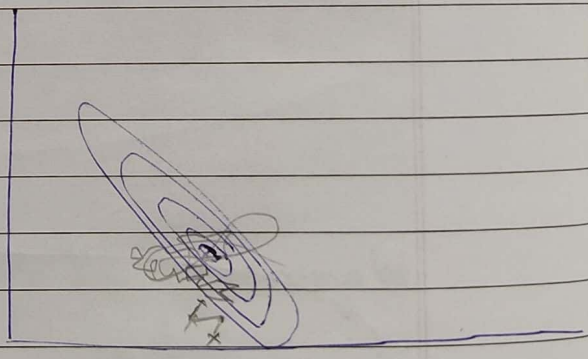
stochastic gd:-
lot like gd but,

- Rather than wait to sum up those gradient terms over all m training e.g., we are taking this gradient term using just one single training e.g., a
And we are starting to make progress in improving parameters already.

Rather than scan through all m training e.g., before this we can modify the parameters a little bit and make progress towards global minimum.
We just need to look at single e.g. we're already starting to make progress. Cmoving the parameters towards global minimum.



Batch GD.



Stochastic GD.

2nd, ita, unlucky, head in bad dir.
3rd, better

Stochastic GD:

- We find, generally move in dir of global minimum, but not always.
- Doesn't converge in same sense as Batch GD does, wandering around continiously in some region close to global minimum, but it doesn't get to global minimum & stay there. → No Problem. We get pretty good hypothesis.

- We get parameter near global minimum & that's good enough.

How many times to run Repeat { loop }
→ Depending on size of dataset, doing this loop just single time may be enough.!!! (1-10 times)

Batch GD:

After taking pass through entire dataset
→ Just one single gradient descent step. (one of baby step)

fast.
SGD > BGD

Q) Statements about Stochastic GD.?

→ ✓. When the training set size m is very large, SGD can be ~~very~~ much faster than BGD.

✓. The cost fⁿ $J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ should go down with every itⁿ of BGD, but not necessarily with SGD.
One e.g. may ↑, & other may ↓, other ↑ (cost) in SGD.

SGD is applicable to LR, Logistic R, NN.

✓. Before beginning of main loop of SGD, its a good idea to shuffle dataset.

Mini Batch Gradient Descent - - - - - - -

sometime may run faster than SGD.

Batch GD: Use all m examples in one each iteration.

Stochastic GD: Use 1 e.g. in each iteration.

Mini batch GD: Use b examples in each iteration.

b = mini-batch-size b=10 usually b=(2-100)

Get b=10 e.g.'s $(x^{(i)}, y^{(i)}), \dots, (x^{(i+9)}, y^{(i+9)})$

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

$i = i + 10;$

☆ 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 $j = 0, \dots, n$)

}

}

$m = 300,000,000$ $b = 10$

looking at just first 10 examples we can start to make progress in improving parameter θ

Second 10 e.g's. ---

MbGD can be faster than BGD.

#MbGD vs SGD. MbGD likely to outperform SGD only if we have a good vectorized implementation.

b e.g's.

1 e.g.

• Parallization (Sum over 10 e.g's can be performed in more vectorized way)

Use libraries to parallelize gradient computations over b examples.

• MbGD disadvantage → Extra parameter. b.

Suppose you use mini-Gradient Descent on training set of size m, & use MbGD size of b. The algo becomes BGD if. $b = m$

-- @ Stochastic Gradient Descent Convergence — — —

#. making sure → converging.
- Picking α
- Debugging.

✡ Checking for convergence.

Batch gradient descent:

- Plot $J_{train}(\theta)$ as a fn of no. of iterations of gradient descent.
- $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$J(\theta)$	↓ on every iteration.

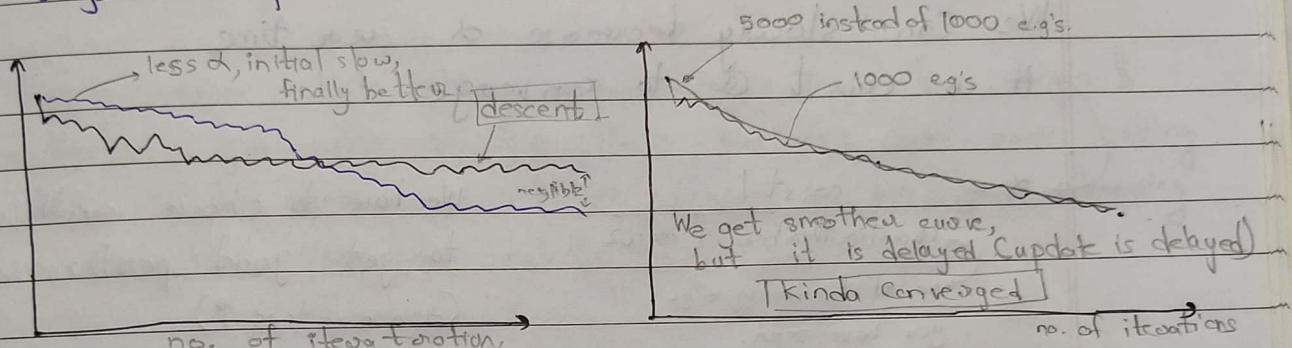
m

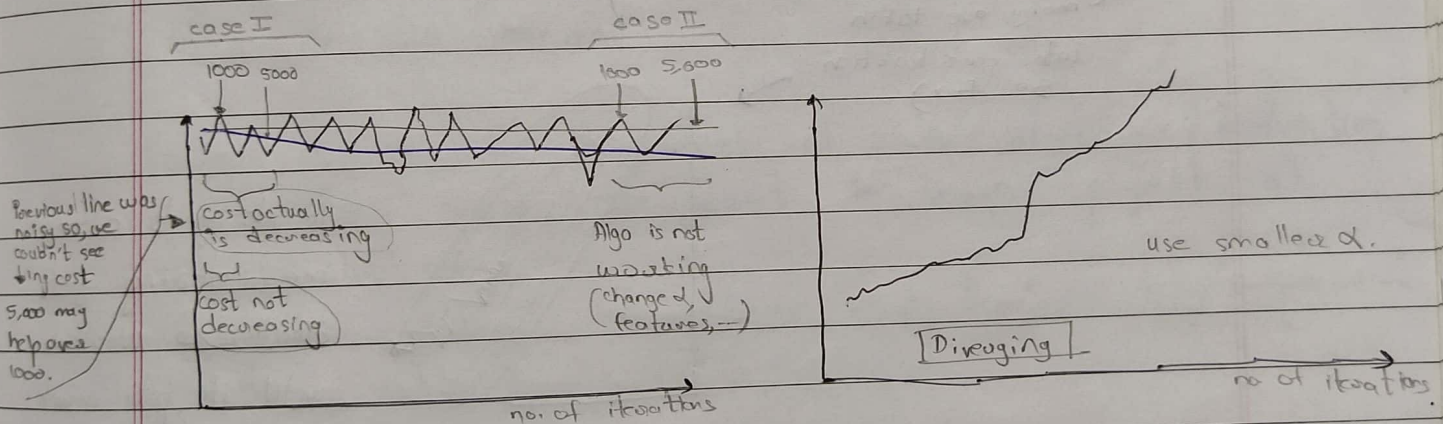
Stochastic Gradient descent:

- $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2 \rightarrow (x^{(1)}, y^{(1)}), (x^{(i+1)}, y^{(i+1)}) \ldots$
- 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 ~~procee~~ processed by algorithm.

Checking for convergence in Stochastic GD.

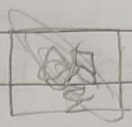
Plot cost $(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples.





Stochastic GD doesn't just converge to global minimum. Parameters oscillate around global minimum. small alpha, you'll end up with small oscillations.

Learning rate α :→



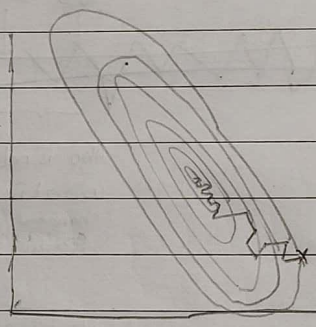
Learning rate α is typically held constant. (We never changed α while algo. is running)

We can slowly decrease α over time if we want α to converge.

$$\alpha = \frac{\text{const1}}{\text{iteration Number} + \text{const2}}$$

iteration Number → no of training e.g. you have seen. (out of m, how many are taken into consideration so far)

We have to choose const1, const2 to get good performance. (#parameters ?)



As its gets closer meandering gets smaller & smaller until its pity much close to global minimum

Alpha get smaller, so steps size ↓'s

But still ~~keepp~~ keeping α const, is more common (const1, const extra parameter)

Choose about stochastic GD.

- Picking a α very small has no disadvantage & can speed up learning.
- If we reduce α (can STOP long enough). Its possible that we may find a set of better parameters ~~wi~~ than with larger α
- If we want SGD to converge to a local minimum rather than wander of "oscillate" around it, we should slowly increase α over time.
- If we plot cost$(\theta, (x^{(i)}, y^{(i)}))$ (averaged over the last 1000 examples) and SDG doesnot seem to be reducing the cost, one possible problem may be that the α is poorly tuned

--@ Online learning — — — —

#.large scale ml setting. Use only if data comming is very large.
• continious stream of data is comming. (We learn & discard ∵ data set is big) assumption.

#online learning.

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

We want to optimize asking price that we want to offer to our users

Feature x capture properties of user, of origin / destination and asking price. We want to learn $p(y=1|x;\theta)$ to optimize price.
↓ includes price we're asking for.

We will use logistic Regn^.

• Algo flow.

Repeat forever {
  Get (x,y) corresponding to new user.
  ~~$(x^{(i)}, y^{(i)})$~~ We learn on e.g. & throw that e.g. and never use it again.
  Update θ using just (x,y)

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

}

• Can Adapt to changing user preference.

e.g.2) Other online learning example.

Product search (learning to search)

User searches for "Android phone 720p cam" → type search
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 discription of phone,)---

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

Learn $p(y=1 \mid x;\theta)$ Estimate probablity that user will click on link of mobile phone.

If y=1, show in search result.

This is called "(The problem of learning the predicted click-through rate, the predicted CTR)"

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

Every time user searches, we show 10 results,
We will get 10 (x,y) pairs & use online learning algo to update parameter using 10 steps of GD on these 10 ((x,y) pairs) e.g.'s & then throw the data away.

other eg.) Choosing special offer to show user; customized selection of news article; product recommendation; ...

If you have collaborative filtering, you can of giving you additional features to feed into logistic regn classifier to try to predict CTR for diff products.

Any of these problem can be formulated as standard ml problem where you have fixed dataset.

Q) Some of advantages of using an online learning algo are

→ ✓ It can adapt to changing user taste ($p(y|x;\theta)$ changes over time)

- There is no need to pick a α.

✓ It allows us to learn ~~from~~ from a continious stream of data, since we use each example once then no longer need to process it again.

- It doesn't require that good features be chosen for learning task.

Similar algo to SGD (training set doesn't change)

JEFF DEAN / Sanjay Ghemawat.

400,000,000 ⟶ 400 (we are using)

Sadguru
Page No: 202
Date: 18/ 9 / 19

--(6) Map reduce and Data Parallelism — — —

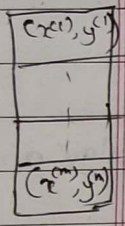
✱ Map reduce:

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

same

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

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



4 machines

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

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

Machine 3: use $(x^{(201)}, y^{(201)}), \ldots, (x^{(300)}, y^{(300)})$

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

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

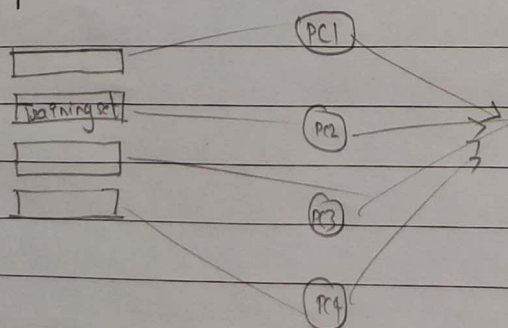
$temp_j^{(4)} = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

combine.

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

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

Map reduce.



(x4 speedup)
↳ 4 PC's.

Map reduce and summation over the training set.

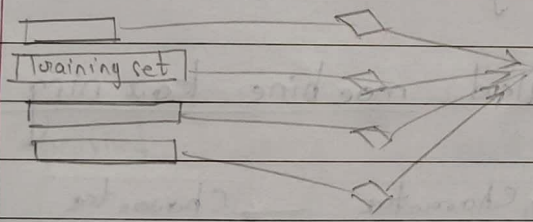
Many learning algorithms can be expressed as computing sums of functions over the training set.

e.g. for advanced optimization, with logistic regn, need:

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

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

Multi-core machines.



Q) Suppose you apply mapreduce method to train a ~~neural~~ neural network on ten machines. In each iteration, what will each of machines do?

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

- Compute either forward prop / back prop on 1/5 of data.
- Compute only FP on 1/10 of data (BP on all data → centralized)
- compute BP on 1/10 of data (FP