Learning with large datasets

Stochastic Gradient Descent

- Linear regression with gradient decent

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

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

Repeat {

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

$j = 0, 1, \ldots, n$

}

$\boxed{\quad}$: Batch gradient decent

Computational expensive since need to load all m examples to do one small step

scanning ~~through~~ the entire dataset peridically

- Stochastic gradient decent

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

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

1) Randomly shuffle dataset

2) Repeat { (usually 1 ~ 10 times repeating, depends on m)

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

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

(for $j = 0, 1, \ldots, n$) $= \frac{\partial}{\partial \theta_j} cost(\theta, x^{(i)}, y^{(i)})$

}

}

Why stochastic gradient descent is much faster than gradient descent?

- for large dataset, gradient decent has to run through all training set to make on one baby step of descent
- Stochastic gradient dexcent take one descent step only when run through one training example. Especially when dataset is large, running through all training examples once is enough to get good training result.

- Mini-batch gradient descent

Batch Gradient Descent uses all m

Stochastic Gradient Descent uses 1

Mini Gradient Descent uses b

} Examples in each iteration

- Mini-batch gradient descent

1) choose b.

2) Repeat {

for $i = 1, 1+b, 1+2b, \cdots$ {

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

(for every $j = 0, 1, \cdots n$)

}

}

- Mini-batch gradient descent can be faster than stochastic gradient descent
  - ☆ vectorization!
  - Mini-gradient descent use b examples for one step, can be vectorized, meaning partially parallelize !!!

Stochastic Gradient Descent Convergence

- Checking for convergence
  - $\text{Cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})$
  - During the learning compute cost $(\theta, (x^{(i)}, y^{(i)}))$ before updating using $(x^{(i)}, y^{(i)})$
  - Every 1000 iterations (say), plot cost averaged over the last 1000 examples processed by the algorithm.

* Stochastic gradient descent does not converge to the global minimum, but only oscillates around the global minimum

* Still need to choose the proper learning rate α.

* if want to SGD to converge, can slowly decrease learning rate α over time

  e.g. $\alpha = \frac{\text{const 1}}{\text{iteration\# + const 2}}$

Online learning

Eg1: Shipping service website. users sometimes choose your shipping service ($y=1$)
Sometimes not ($y=0$)
Features x capture properties of users, want to learn $P(y=1|x;\theta)$
to optimize the prize to get maximum users

Background: have large stream of user's data

Repeat {
    Get (x,y) corresponding to user
    update θ using (x,y)
        $\theta_j := \theta_j - \alpha(h_\theta(x)-y)\cdot x_j$ $(j=0,1,\ldots,n)$
    Discard (x,y) after using it once
}

* Advantage of Online learning
    - Can adapt to change of users' preferences

Eg 2: 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 that matches the query

y: 1 if users clicked ~~the~~ button

0 ~~otherwise~~

learn $P(y=1|x;\theta)$ [predicted CTR]

CTR: Click through rate

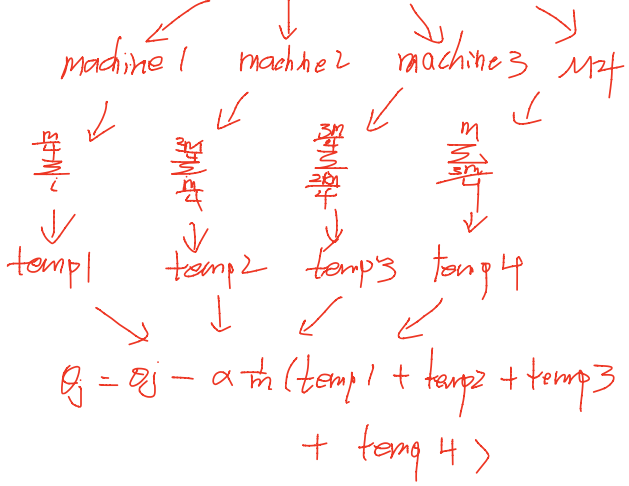
other examples: choosing special offers ~~to~~ users; customized of selection of news articles; product recommendation

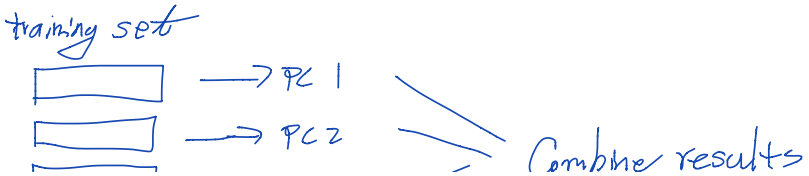
These examples can also be ~~transfer to~~ classical ML problem

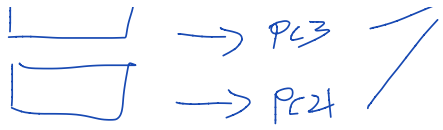
Map-reduce and data parallism

- Map-reduce

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







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

- Multi-cores machines

  pc i => core i