Large Scale Machine Learning

Learning with Large Data Sets

$m = 100{,}000{,}000$ Why not $m = 1{,}000$? Sanity check!

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

Sanity check $m = 1000$

high variance learning curves

Stochastic Gradient Descent

linear regression w/ grad 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 {

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

$m = 300{,}000{,}000$

Batch gradient descent SLOW

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

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

}

Stochastic GD

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

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

1. Randomly shuffle dataset
2. Repeat { 1-10 times

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

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

$(\text{for } j=0,\dots,n) \quad \underbrace{}\frac{\partial}{\partial\theta_j} Cost(\theta,(x^{(i)},y^{(i)}))$

}

}

$(x^{(1)},y^{(1)}), (x^{(2)},y^{(2)})$
1st → 2nd .....

Mini-Batch Gradient Descent (sometimes faster than stochastic)

Batch: use all m examples in each iteration
Stochastic gradient descent: Use 1 example "
Minibatch: Use b examples "
$\qquad\qquad\qquad 2\text{-}100$

b = mini-batch size $\qquad b=10$
Get $b=10$ examples $\quad (x^{(i)},y^{(i)}),\dots(x^{(i+9)},y^{(i+9)})$

for $i=1,11,21\dots\{ \theta_j := \theta_j - \alpha\frac{1}{10}\sum_{k=1}^{i+9}(h_\theta(x^{(k)})-y^{(k)})x_j^{(k)}$

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

}

b examples vs. 1 example

Vectorization

disadvantage new parameter b

Stochastic Gradient Descent Convergence

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

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

Every 1000 iterations (say), plot $cost(\Theta,(x^{(i)},y^{(i)}))$ averaged over the last 1,000 examples processed by algo

5,000 for flatter lines / if too noisy

Learning rate α is typically held constant. CAN decrease α over time if we want Θ to converge

(E.g. $\alpha = \frac{const1}{iteration\# + const2}$)

Online Learning (streaming)

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

Repeat forever {

Get (x,y) corresponding to user.

Update Θ using (x,y):

$$\Theta_j := \Theta_j - \alpha(h_\theta(x)-y)\cdot x_j \qquad (j=0,\dots,n)$$

} → Can adapt to changing user preference.

Predicted Click-through-rate (CTR)

Map Reduce & Data Parallelism

M/R $m=400$ $m=400{,}000{,}000$

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

4 Machines

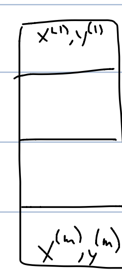
Machine 1: Use $(x^{(1)},y^{(1)}),\dots,(x^{(100)},y^{(100)})$

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

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

$$temp_j^{(2)} = \sum_{i=1}^{100}(h_\theta(x^{(i)})-y^{(i)})\cdot x_j^{(i)} \longrightarrow$$

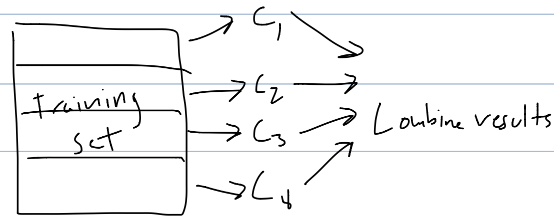
⋮



master server

Combine:

$$\theta_j := \theta_j - \alpha \frac{1}{400}\left(temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)}\right) \quad (j = 0, \dots, n)$$



Multi-core Machines