Week 10

- ## Gradient Descent with Large Datasets:
  a. ### Learning With Large Datasets:
  Before we use massive data sets, we can check if one thousand is ok.
  The way to check of using a much smaller training set: plotting the learning curve.

  b. ### Stochastic Gradient Descent:

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

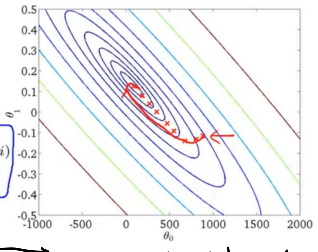
$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Repeat {

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

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

}



m = 300,000,000

Batch gradient descent refers to that we're looking at all of the training examples at a time.

Batch gradient descent

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

Repeat {

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

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

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

}

m = 300,000,000

Stochastic gradient descent

$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$ just a single example.

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

1. Randomly shuffle dataset.
2. Repeat { reorder.

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

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

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

}

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

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

  c. ### 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

b = mini-batch size. b = 10. 2 - 100

Get b = 10 examples $(x^{(i)}, y^{(i)}), \ldots (x^{(i+9)}, y^{(i+9)})$

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

$i := i + 10$

Say $b = 10, m = 1000.$

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

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

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

}

}

  d. ### Stochastic Gradient Descent Convergence:
  How do you make sure that the algorithm is completely debugged and is converging okay? And how to tune the learning rate.

Checking for convergence

→ Batch gradient descent:

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

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

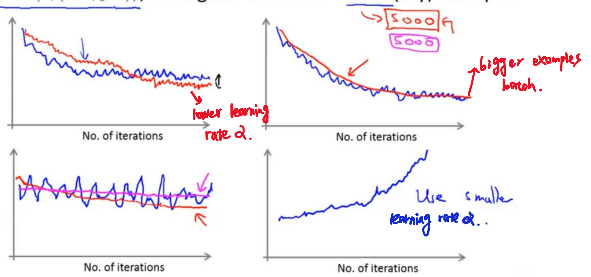
→ Stochastic gradient descent:

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

→ 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 algorithm.

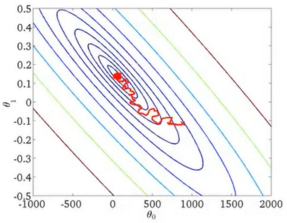
Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples



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

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

1. Randomly shuffle dataset.
2. Repeat {
   for $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }



Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{const1}{iterationNumber + const2}$) $\alpha \to 0$

• Advanced Topics:

a. Online Learning:

Online learning

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

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

price logistic regression

Repeat forever {
Get (x, y) corresponding to users. $(x^{(i)}, y^{(i)})$.
update θ using (x, y).
$\theta_j := \theta_j - \alpha(h_\theta(x) - y) \cdot x_j$
}.
Can adapt to changing users preference.

ther online learning example:

oduct search (learning to search)
User searches for "Android phone 1080p camera"
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

b. Map Reduce and Data Parallelism:

Map-reduce

Batch gradient descent: $\theta_j := \theta_j - \alpha\frac{1}{400}\sum_{i=1}^{400}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ $m = 400$ $m = 400,000,000$

Machine 1: Use $(x^{(1)}, y^{(1)}), \ldots, (x^{(100)}, y^{(100)})$. $temp_j^{(1)} = \sum_{i=1}^{100}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Machine 2: Use $(x^{(101)}, y^{(101)}), \ldots, (x^{(200)}, y^{(200)})$. $temp_j^{(2)} = \sum_{i=101}^{200}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

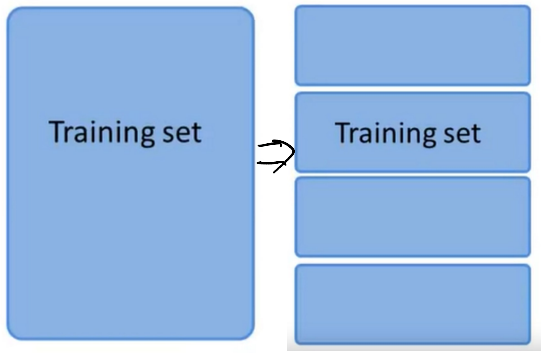
Machine 3: Use $(x^{(201)}, y^{(201)}), \ldots, (x^{(300)}, y^{(300)})$. $temp_j^{(3)} = \sum_{i=201}^{300}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

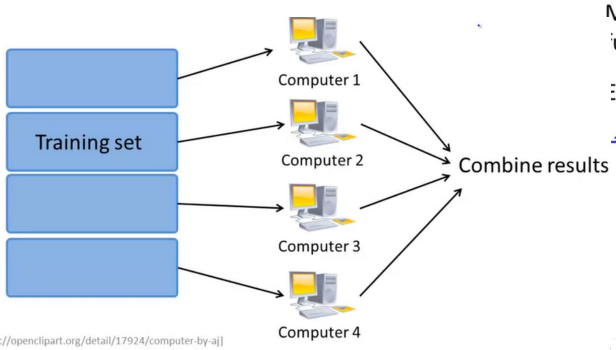
Machine 4: Use $(x^{(301)}, y^{(301)}), \ldots, (x^{(400)}, y^{(400)})$. $temp_j^{(4)} = \sum_{i=301}^{400}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$

Combine:

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

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





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

E.g. for advanced optimization, with logistic regression, need:

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

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

$temp_j^{(i)}$ $temp_j^{(i)}$ ←