Machine Learning - Week 10 - Large Scale Machine Learning

T1 - Gradient Descent with Large Datasets

L1 - Learning with large datasets

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 of the data is likely to perform much better than using a small subset of the data (say $m = 1000$)?

-> Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.

L2 - Stochastic Gradient Descent

Recall: Batch gradient descent:

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

Repeat {

$$\theta_j := \theta_j - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}}_{\frac{\partial}{\partial\theta_j} J_{train}(\theta)}, \quad (\text{for every } j = 0, \ldots, n)$$

}

Stochastic gradient descent:

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

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

Steps: 1) Randomly shuffle dataset

2) Repeat { (1-10x)

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

$$\theta := \theta_j - \alpha \underbrace{\left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)}}_{\frac{\partial}{\partial\theta_j} cost(\theta, (x^{(i)}, y^{(i)}))}, \quad (\text{for } j = 0, \ldots, n)$$

}

}

L3 - Mini-batch gradient descent

Recall:

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 (usually 2 - 100)

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

Repeat {

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

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}, \quad (\text{for every } j = 0, \dots, n)$$

}

}

L4 - Stochastic Gradient descent convergence

Checking for convergence.

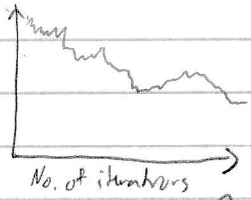
Recall: Batch Gradient descent:

- Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.
- $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

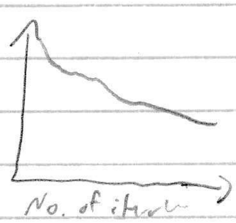
Stochastic gradient descent:

- $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$
- 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 1000 examples processed by algorithm

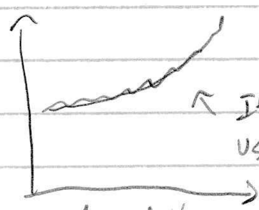
Checking for convergence



avg over 1000 examples



avg over 5000 examples



If diverging, use smaller α

Learning rate α is typically held constant. Can slowly decrease α over time if we want Θ to converge (Eg. $\alpha = \frac{const1}{iterationNum + const2}$)

T2 - Advanced Topics

L1 - Online Learning

Ex. 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 $p(y=1 \mid x; \theta)$ to optimize price.

↑includes price

Repeat Forever {

Get (x, y) corresponding to user.

Update Θ using (x, y):

$$\Theta_j := \Theta_j - \alpha (h_\Theta(x) - y) \cdot x_j \quad (j = 0, \ldots, n)$$

}

L2 - MapReduce and Data Parallelism

Map-Reduce

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

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