Stochastic Gradient Descent

Sid

- alternative to batch gradient descent
- more efficient / scalable for large data sets

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

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

1) Randomly shuffle dataset

2) Repeat { 1 training example @ a time $(x^{(i)}, y^{(i)})$

for $i = 1 : m$ {

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

(for every $j = 0 \rightarrow n$)

Mini-Batch Gradient Descent:

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

Stochastic Gradient Descent Convergence

- plot avg cost vs training examples
- oscillates around global minimum : smaller α may give smaller cost
- Can slowly decrease α : $\dfrac{\text{const}1}{\text{iter}\# + \text{const}2}$

Online Learning

Repeat forever {

Get (x, y) — user

Update θ using (x, y)

↳ $\theta_j := \theta_j - \alpha \left(h_\theta(x) - y\right) x_j$

}

☆ Can adapt to user preference change

☆ learn from continuous stream of data (discard x, y)

Map-reduce & Data Parallelism

- Divide batch gradient descent and dispatch cost function for subset of data to train algorithm in parallel (w/ different machines)
- training set → z subsets : $\sum_{i=p}^{q} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)}$

Map-Reduce : $\theta_j := \theta_j - \alpha \frac{1}{z} \left(temp_j^{(1)} + \ldots + temp_j^{(z)}\right)$

Map-Reduceable if algorithm can be expressed as computing sums of functions over training set (linear, logistic, ...) ✓