Large Scale Machine Learning

Sidharth Baskaran

June 2021

Learning with Large Datasets

- Gradient descent with large m is computationally expensive
- High variance with small m indicates that large dataset is necessary, however
- Plot learning curve with J_{CV} and J_{train}
  - Large difference indicates **high variance**, else a large m does not make much of a difference

Stochastic Gradient Descent

- Allows for upscaling, modification of original gradient descent
- Computing partial derivative is computationally expensive
- Batch gradient descent $\rightarrow$ consider all m training examples at once in update routine
  - $J_{\text{train}}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$
  - Repeat $\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ for every $j = 0, \ldots, n$
- Stochastic gradient descent
  - Define $\text{cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$
  - $J_{\text{train}} = \frac{1}{m}\sum_{i=1}^{m}\text{cost}(\theta, (x^{(i)}, y^{(i)}))$
  - Randomly shuffle training examples
  - Repeat **(1-10x)** (for $i \in (1, \ldots, m)$)
    - $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)} - y^{(i)}))x_j^{(i)}$
      - Do not need to sum up all training example partial derivatives
    - For all $j \in (0, \ldots, n)$
  - Each iteration of SGS is fast, and will generally **but not always** move towards global minimum
    - Wanders around continuously

Mini-batch gradient descent

- Batch GD $\rightarrow$ all m exampels in each iteration
- Stochastic GD $\rightarrow$ use 1 example in each iteration
- Mini-batch GD $\rightarrow$ use $b \ll m$ examples in each iteration
  - Ex: get $b = 10$ examples, then perform gradient descent update $\theta_j := \theta_j - \alpha\frac{1}{10}\sum_{k=i}^{i+9}(h_\theta(x^{(k)}) - y^{(k)})x_j^{(k)}$ for all $j \in (1, \ldots, n)$
  - Then, $i := i + 10$

Stochastic gradient descent convergence

- Batch Gradient Descent
  - Plot $J_{\text{train}}(\theta)$ over iterations of GD
- Stochastic GD
  - During learning, compute $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ prior to updating θ using $x^{(i)}, y^{(i)}$

  - Can plot cost averaged over last 1000 iterations every 1000 iterations or so to adjust α if necessary
  - Can slowly decrease α over time $\rightarrow a := \frac{\text{const1}}{\text{iterNum} + \text{const2}}$ in order for θ to converge

Online learning

- Continuous input stream of data
- Example $\rightarrow$ shipping service, where user is offered asking price
  - User choosing service is $y = 1$ and not is $y = 0$
  - Features x provide information about user, origin/dest, asking price
  - Want to learn $p(y = 1|x; 0)$ to optimize price
- Algorithm
  - Repeat forever
  - Get (x, y) pair representing a user
  - Update θ using only this example
    - $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j, \forall j$

Map Reduce and Data Parallelism

Suppose

$$\text{training set} = \begin{bmatrix} (x^{(1)}, y^{(1)}) \\ \vdots \\ - - - - \\ \vdots \\ - - - - \\ \vdots \\ - - - - \\ \vdots \\ (x^{(i)}, y^{(i)}) \end{bmatrix}$$

- First machine uses 1st k examples from divided set
  - $\text{temp}_j^{(i)} = \sum_{i=1}^{k}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
  - And so on for each machine assigned to subset
- Send all temp variables to master server and update $\theta_j := \theta_j - \alpha\frac{1}{m}(\text{temp}_j^{(1)} + \ldots + \text{temp}_j^{(m/k)})$
  - Can get almost m/k-times speedup
- Same as regular batch gradient descent, but benefit of parallelism
- Many learning algorithms can be expressed as computing sums of functions over training set, so parallelism is straightforward
- Multi-core machines
  - Distribute workload across multiple cores