Overview

- Previously we have seen how to learn a linear regression model using distributed closed-form solution.
- This distribution solution runs into minor issues when we try to scale.

Limitations of the Closed-Form OLS: Solving for the θs

- Problem is equivalent to inverting `X'X` matrix.
    - Inverse does not exist if matrix is not of full rank.
        - E.g., if one column is a linear combination of another (collinearity).
        - Note that `X'X` is closely related to the covariance of the `X` data.
            - So we are in trouble if two or more variables are perfectly correlated.
        - Numerical problems can also occur if variables are almost collinear.
- Equivalent to solving a system of *p* linear equations.
    - Many good numerical methods for doing this, e.g., Gaussian elimination, LU decomposition, etc.
    - These are numerically more stable than direct inversion.
- Matrix inversion is not easily parallelized.

Alternative Ways of Solving OLS

- Closed-form solution
  - In this method, we will minimize RSS by explicitly taking its derivatives with respect to the β_j's (sometimes written as W, the weight vector) and setting them to zero.
  - Do this via calculus with matrices.

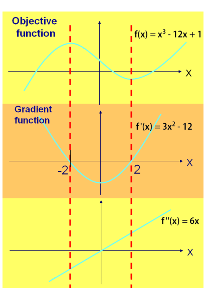
$\theta^* = (X^T X)^{-1} X^T y$ →

- Gradient descent gives another way of minimizing RSS.
- Bayesian approach.
- Quadratic programming.
- Others.

Linear Regression via Gradient Descent

Given: Minimize *f(x)*. $J_\theta(W, X_1^m) =$ Minimize $\sum_{i=1}^{m} (W^T X_i - y_i)^2$

Step 1: Find the zeros of the gradient function *f'(x)* using bisection method, Newton-Raphson, or gradient descent.



RSS = Variance of ε

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial W} = \frac{\partial \left(\sum_{j=1}^{n} \left(X_j W - y_i \right)^2 \right)}{\partial W}$$

$$\nabla J(W) = \left(\sum_{j=1}^{n} \left(X_j W - y_i \right) X_j \right)$$

Gradient vector of partial derivatives

Pull model closer to examples with biggest residual.

$$W^{t+1} = W^t - \alpha^t \left(\sum_{j=1}^{n} \left(X_j W^t - y_i \right) X_j \right)$$

Gradient descent

For another derivation see: http://cs229.stanford.edu/notes/cs229-notes1.pdf.

OLS Using Gradient Descent

$J_q(W, X_1^m) = \text{Minimize} \sum_{i=1}^{m} (W^T X_i - y_i)^2$ OLS objective function with decision variables *W*

- Initialize *W* = vector or zeros.
- Repeat until convergence.

$W_{t+1} = W_t - \alpha_i \left(\sum_{j=1}^{n} (X_j W_t - y_i) X_j \right)$ OLS batch update rule

- End repeat.

True gradient is approximated by the gradient of the cost function only evaluated at all examples; adjust parameters proportional to this approximate gradient.

Intuitively, drag weight vector closer to the incorrectly predicted examples.

Batch vs. Stochastic

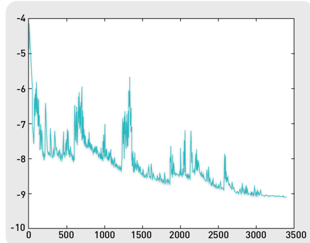
- That was batch-based descent.
- Stochastic gradient descent vs. batch.

OLS Using Gradient Descent

Stochastic gradient descent

∇ J wj (W t)

Partial derivative WRT to variable w j of error function J(W) at point W t



Fluctuations in the Total Objective

Stochastic update (after each example)

Let *W* = (0, 0,…)

Repeat

For *j* in 0…*n* #each variable

For *i* in 1…*m* #each example

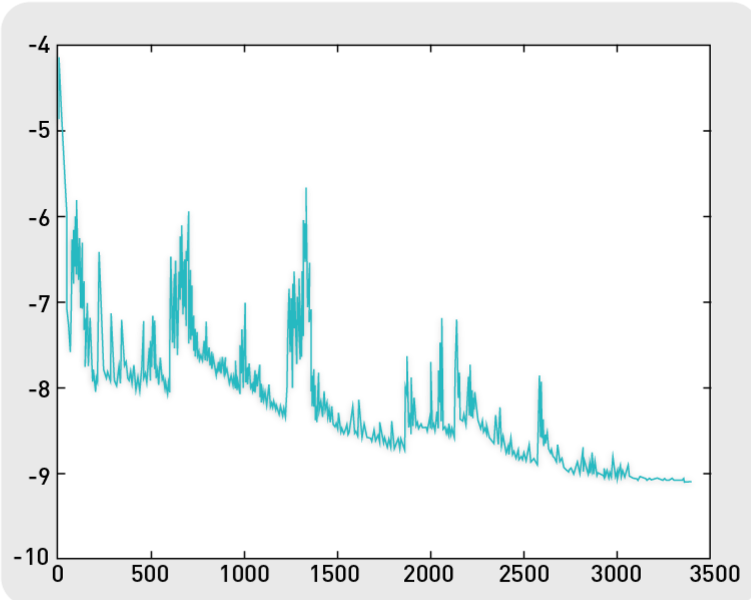
$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^{n} \left(X_j W^t - y_i \right) X_j \right)$$

Until convergence (i.e., no big changes in *W* or error)

Stochastic Gradient Descent vs. Batch

- Stochastic gradient descent can start making progress right away and continues to make progress with each example it looks at.
- Often, stochastic gradient descent gets W "close" to the minimum much faster than batch gradient descent.
    - Note, however, that it may never "converge" to the minimum, and the parameters W will keep oscillating around the minimum of J(W); but in practice most of the values near the minimum will be reasonably good approximations to the true minimum.
- For these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

Stochastic Updates -> Fluctuations in the Total Objective



Fluctuations in the Total Objective

Two Approaches

- Two gradient-descent approaches to linear regression
  - Batch gradient descent
  - Stochastic gradient descent based
- Are both amenable to parallelization? And if so, which do you expect to perform better?

Stochastic Gradient Descent: Not Easy in MapReduce

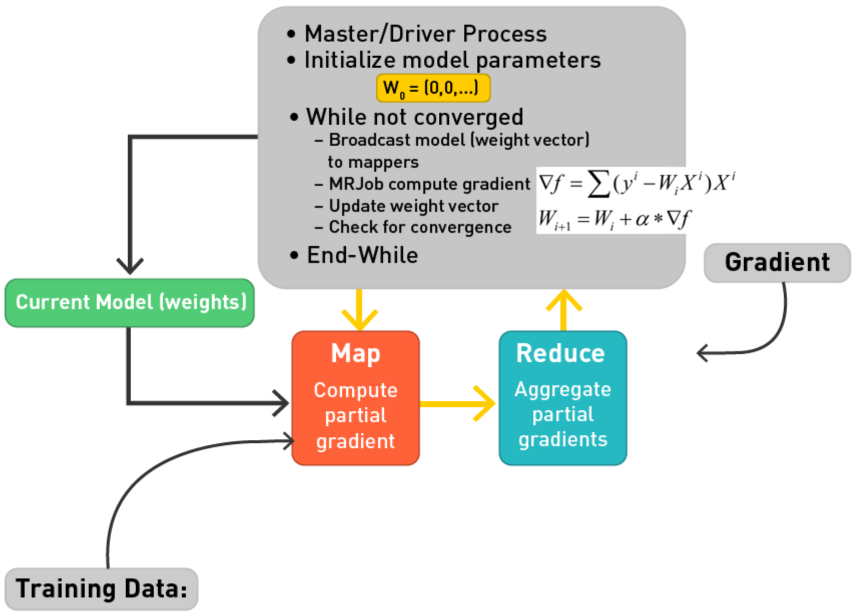
- Stochastic gradient descent requires the weight vector to be updated after each example (or minibatch of examples).
- "Some implementations of machine-learning algorithms, such as ICA, are commonly done with stochastic gradient ascent, which poses a challenge to parallelization. The problem is that in every step of gradient ascent, the algorithm updates a common set of parameters."

Some implementations of machine-learning algorithms, such as ICA, are commonly done with stochastic gradient ascent, which poses a challenge to parallelization. The problem is that in every step of gradient ascent, the algorithm updates a common set of parameters (e.g., the unmixing *W* matrix in ICA). When one gradient ascent step (involving one training sample) is updating *W*, it has to lock down this matrix, read it, compute the gradient, update *W*, and finally release the lock. This "lock-release" block creates a bottleneck for parallelization; thus, instead of stochastic gradient ascent, our algorithms above were implemented using batch gradient ascent.

OLS via Distributed Gradient Descent

- Master/Driver process.
- Initialize model parameters $W_0 = (0,0,\ldots)$.
- While not converged:
  - Broadcast model (weight vector) to mappers.
  - MRJob compute gradient $\nabla f = \Sigma\ (y_i - W_i X_i)\ X_i$.
  - Update weight vector $W_{i+1} = W_i + \alpha * \nabla f$.
  - Check for convergence.
- End-While.

MapReduce Implementation



OLS via Distributed Gradient Descent: Mapper and Reducer

- Master/Driver process.
- Initialize model parameters, *W*= vector of zeros: $W_0 = (0,0, ...)$.
- While not converged:
  - Broadcast model (i.e., weight vector) to the worker nodes.
  - Mapper (MANY mappers).
    - Compute partial gradient for each training example.
    - Combine in memory $\nabla f = \Sigma (y^i - W_i X^i) X^i$.
    - Finally yield the partial gradient.
  - Reducer (single Reducer).
    - Aggregate partial gradients.
    - Yield full gradient $\nabla f = \Sigma (y^i - W_i X^i) X^i$.
  - Update weight vector $W_{i+1} = W_i + \alpha * \nabla f$.
  - Check for convergence.
- End-While.

Divide and Conquer

- Can we use a combiner? Yes!
    - In-memory combiner.
    - So use mapper final.
- How many reducers?
    - We can have one (should be sufficient).
    - Or many (in the case of many, the master has to aggregate the partial reducer aggregates).