

Distributed Logistic Regression Report

Peng Rui, 14331226, School of Data and Computer Science, Sun Yat-Sen University

Concepts and Techniques

Simplified Cost Function

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))]$$

Gradient Descent

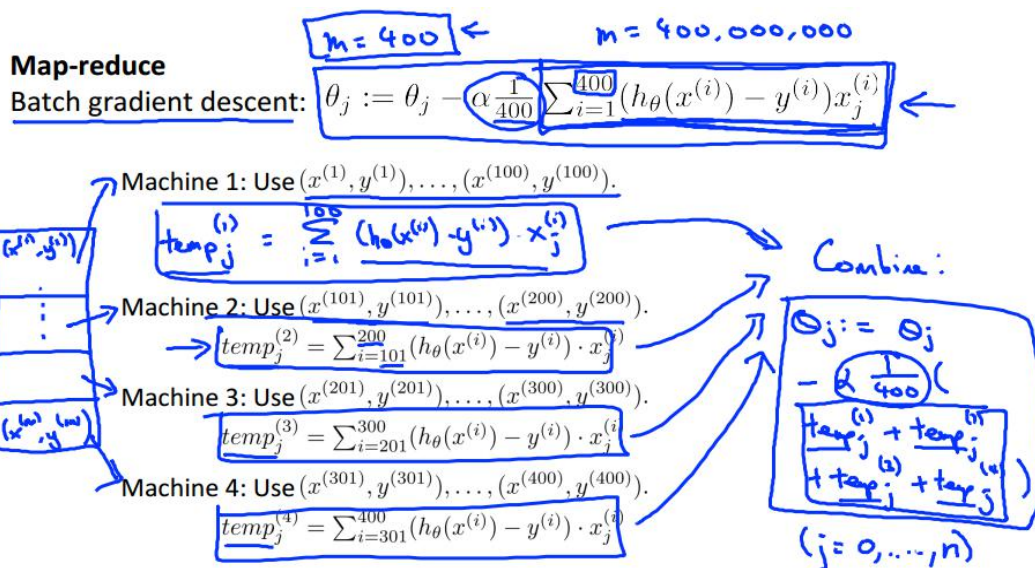
Repeat {

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

}

Map-reduce and data parallelism

We can divide up batch gradient descent and dispatch the cost function for a subset of the data to many different machines so that we can train our algorithm in parallel.



[Jeffrey Dean and Sanjay Ghemawat]

Andrew Ng

Simulating Distributed Logistic Regression

Implementing Cost Function and Gradient Using Vectorization

```
function [J, grad] = costFunctionBatch(m, theta, X, y)
```

```
    activated = sigmoid(X * theta);
    J = - (1 / m) * sum(y .* log(activated) + (1 - y) .* log(1 - activated));
    grad = (1 / m) * X' * (activated - y);
```

```
end
```

Distributing Cost Function and Gradient

`parfor loopVar = initVal:endVal; statements; end` executes for-loop iterations in parallel on workers in a parallel pool.

MATLAB® executes the loop body commands in statements for values of `loopVar` between `initVal` and `endVal`. `loopVar` specifies a vector of integer values increasing by 1. If you have Parallel Computing Toolbox™, the iterations of statements can execute on a parallel pool of workers on your multi-core computer or cluster. As with a for-loop, you can include a single line or multiple lines in statements.

Here, Parallel Computing Toolbox is used to simulate distributed logistic regression.

```
function [J, grad] = costFunctionDistributor(k, m, theta, X, y)

J = 0;
grad = zeros(size(theta));

% parallel computing of the cost function and gradient
parfor i = 1:k
    [Jtemp, Gtemp] = costFunctionBatch(m, theta, X(:, :, i), y(:, i));
    J = J + Jtemp;
    grad = grad + Gtemp;
end

















end
```

Test with Simplified MNIST Database

```
% Batch gradient descent
tic;
[theta1, cost1] = fminunc(@(t)(costFunctionBatch...
    (60000, t, Xtrain, ytrain)), init_theta, param);
pred1 = sigmoid(Xtest * theta1) >= .5;
acc1 = sum(pred1 == ytest) / 10000;
t1 = toc;

% Distributed gradient descent
tic;
Dx = zeros(15000, size(Xtrain, 2), 4);
Dy = zeros(15000, 4);
for i = 1:4
    Dx(:, :, i) = Xtrain(15000 * i - 14999:15000 * i, :);
    Dy(:, i) = ytrain(15000 * i - 14999:15000 * i);
end
[theta2, cost2] = fminunc(@(t)(costFunctionDistributor...
    (4, 60000, t, Dx, Dy)), init_theta, param);
pred2 = sigmoid(Xtest * theta2) >= .5;
acc2 = sum(pred2 == ytest) / 10000;
t2 = toc;
```

After running both version of logistic regression, we can conclude that the results of both method are very similar, with the testing accuracy of about 0.7813 and cost about 0.4924. However, the distributed version cost more time. Running logistic regression without paralleling costs only 15.7102 seconds, while the same algorithm with paralleling costs 84.5736 seconds.

	acc1	0.7813
	acc2	0.7813
	cost1	0.4924
	cost2	0.4924
	Dx	15000x52x4 double
	Dy	15000x4 double
	i	4
	init_theta	52x1 double
	param	1x1 struct
	pred1	10000x1 logical
	pred2	10000x1 logical
	t1	15.7102
	t2	84.5736
	test	10000x785 double
	theta1	52x1 double
	theta2	52x1 double

Reasons of the Worse Efficiency of Distributed Logistic Regression

Vectorizing code allows you to benefit from the built-in parallelism provided by the multithreaded nature of many of the underlying MATLAB libraries. However, if you have vectorized code and you have access only to local workers, then *parfor*-loops may run slower than *for*-loops. Do not devetorize code to allow for *parfor*; in general, this solution does not work well.

Here, my implementation of the cost function and gradient are vectorized, and the vectorized implementation benefits from the built-in parallelism. And this is probably the reason why my parallel version of logistic regression runs much slower. Also devetorizing to allow *parfor* is deprecated. Vectorizing code in MATLAB helps to make it run faster.

Conclusion

The materials and references of this report are mainly from the course [Machine Learning](#) and [MATLAB documentation](#). A learning algorithm is Map-reduceable if it can be expressed as computing sums of functions over the training set. Linear regression and logistic regression are easily parallelizable.

However, in MATLAB, vectorized code has already benefited from the built-in parallelism provided by the multithreaded nature of many of the underlying MATLAB libraries. As a result, if you want to make code run faster, first try to vectorize it.

Appendix

To reduce the size of the submitted zip flie, the data sets of this project are not included. You can download it from [the MNIST Database](#). The C++ code of initializing the database and MATLAB code of running this algorithm are provided in the src folder.