IMPRO3

Logistic Regression

26.05.2014

Markus Holzemer
Jonas Traub
Timo Walther

Agenda

Motivation

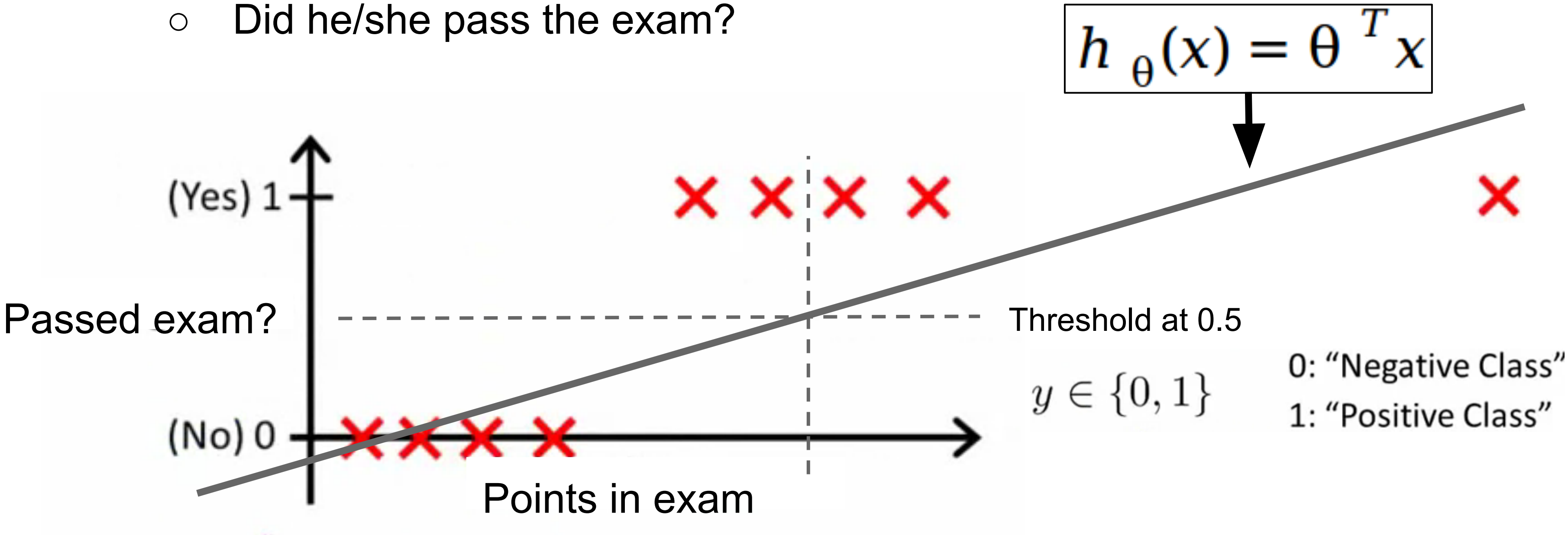
Hypothesis and Cost Representation

Pseudocode
- Batch gradient descent
- Stochastic gradient descent

Parallelization
- Comparison of parallelizability
- Parallel batch gradient descent

Motivation

- Logistic Regression is for **Classification**
- Typically binary classification
  - Is this mail spam?
  - Did he/she pass the exam?



- In example: $h_\theta(x) < 0$ *and* $h_\theta(x) > 1$ *are possible*
- With Logistic Regression: $0 \leq h_\theta(x) \leq 1$

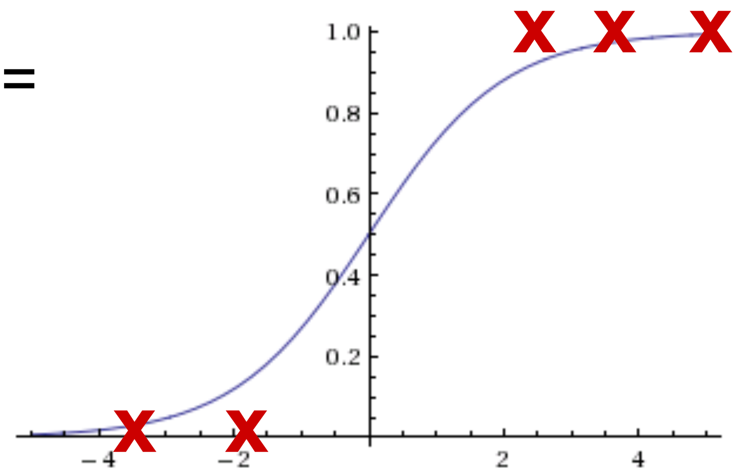
source: coursera/Stanford Machine Learning by Andrew Ng

Hypothesis and Cost Representation

Sigmoid Function = Logistic Function =

$$g(z) = \frac{1}{1+e^{-z}} \quad \text{with } h_\theta(x) = g(\theta^T x)$$

$$\Rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



wolframalpha.com: plot g(z)=(1/(1+e^(-z))) from z = -5 to +5

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) \qquad Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1-h_\theta(x)) & \text{if } y = 0 \end{cases}$$

=> We want to minimize cost J

=> Gradient Descent, repeat:

$$\theta_j = \theta_j - \alpha \frac{\Delta J(\theta)}{\Delta \theta_j} \text{ with } \frac{\Delta J(\theta)}{\Delta \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$$

Pseudocode

```
X = [m, n]        // training set of features
y = [m]           // vector of classification
alpha = 1         // learning rate
theta = [n] -> all 0

Gradient descent:
for 1:number_iterations
    for i = 1:n
       grad(i) = 0;
       for j = 1:m
          grad(i) += (sigmoid(X(j,:)*theta)-y(j))*X(j,i));
       end
       grad(i) = grad(i)/m;
    end
    theta = theta - alpha * grad;
end
```

derivative of cost function

h(x)

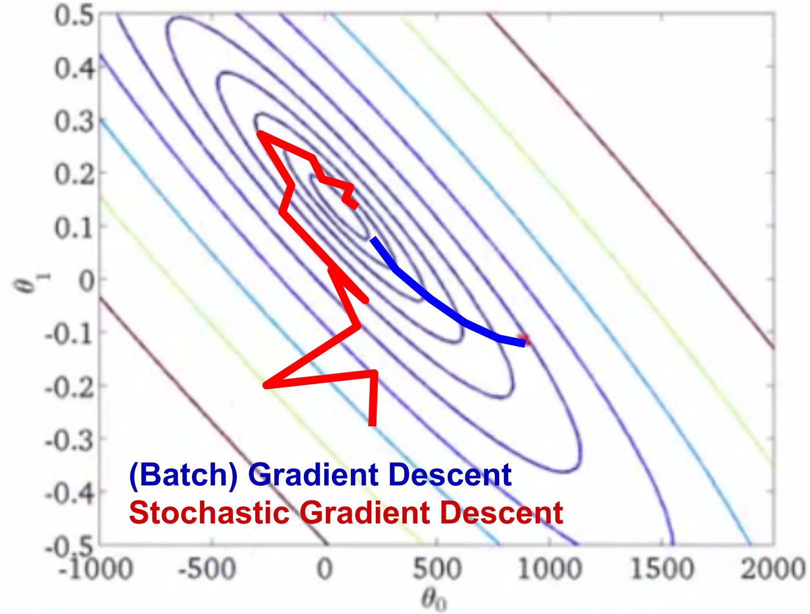
Very naive way,
can be vectorized

Stochastic Gradient Descent (for large training sets)

```
X = [m, n]       // training set of features
y = [m]          // vector of classification
alpha = 1        // learning rate
theta = [n] -> all 0

Stochastic Gradient Descent:

Randomly_Shuffle_Training_Set(X,y)
repeat until theta converges
    for j = 1:m
        for i = 1:n
            grad(i) = (sigmoid(X(j,:)*theta)-y(j))*X(j,i));

        end
        theta = theta - alpha * grad;
    end
end
```

different cost function but same derivative

h(x)



=> make progress in each iteration (modify the parameters to fit the training set a little bit better)

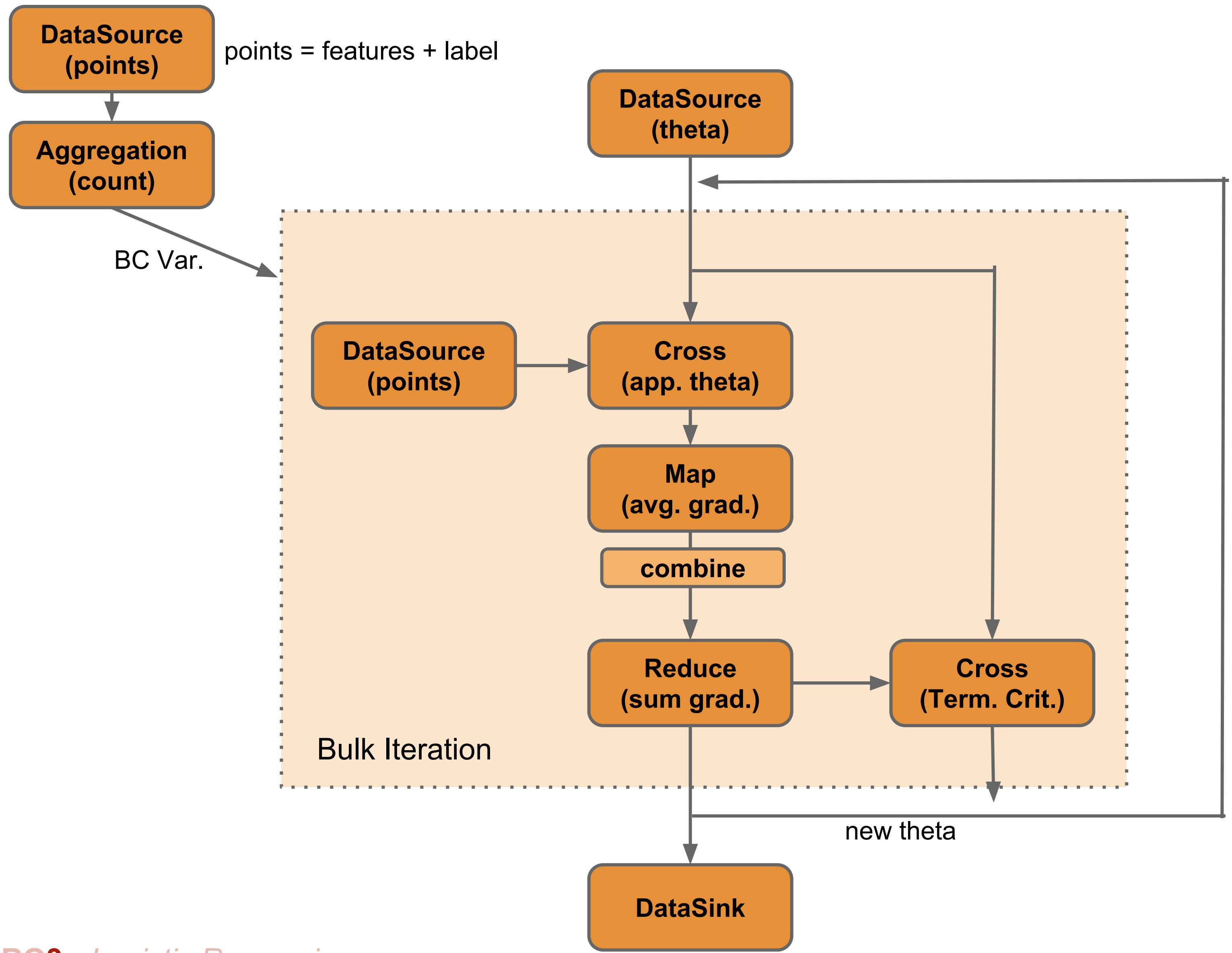
=> generally, move the parameters in the direction of the global minimum

Parallelization

- **Stochastic Gradient Descent**
  - Inherently not parallelizable (theta needs to be adjusted after every point)
  - Parallelization over different alphas or different distributions of the training set?

- **Batch Gradient Descent**
  - Parallel computation of the average gradient over all points possible (see next slide)
  - But: Not clear if it is profitable in comparison to a local SGD

Parallel Batch Gradient Descent



Questions?