

Gradient Descent

Mohammad Emtiyaz Khan
EPFL

Sep 22, 2015



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

©Mohammad Emtiyaz Khan 2014

Learning/estimation/fitting

Given a cost function $\mathcal{L}(\boldsymbol{\beta})$, we wish to find $\boldsymbol{\beta}^*$ that minimizes the cost:

$$\min_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}), \quad \text{subject to } \boldsymbol{\beta} \in \mathbb{R}^{D+1}$$

This is learning posed as an [optimization](#) problem. We will use an [algorithm](#) to solve the problem.

Grid search

Grid search is one of the simplest algorithms where we compute cost over a grid (of say M points) to find the minimum.

This is extremely simple and works for any kind of loss when we have very few parameters and the loss is easy to compute.

For a large number of parameters, however, grid search has too many “for-loops”, resulting in exponential computational complexity. Choosing a good range of values is another problem.

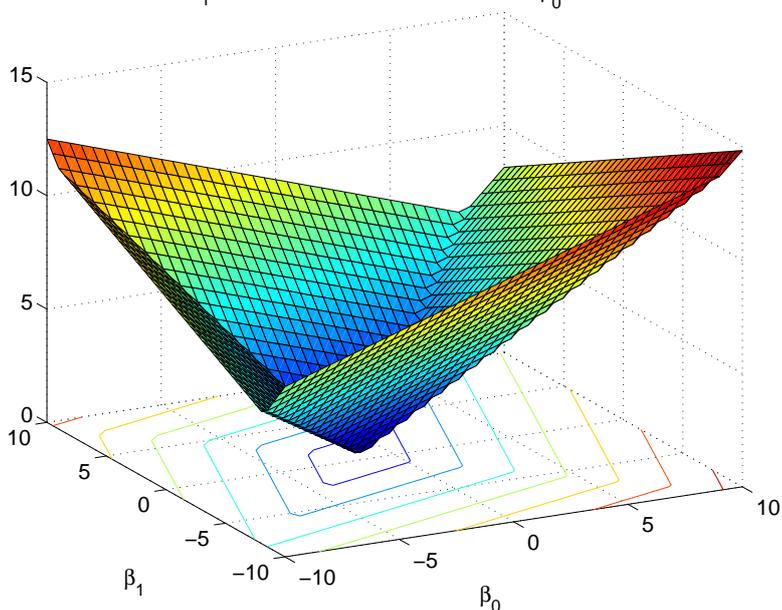
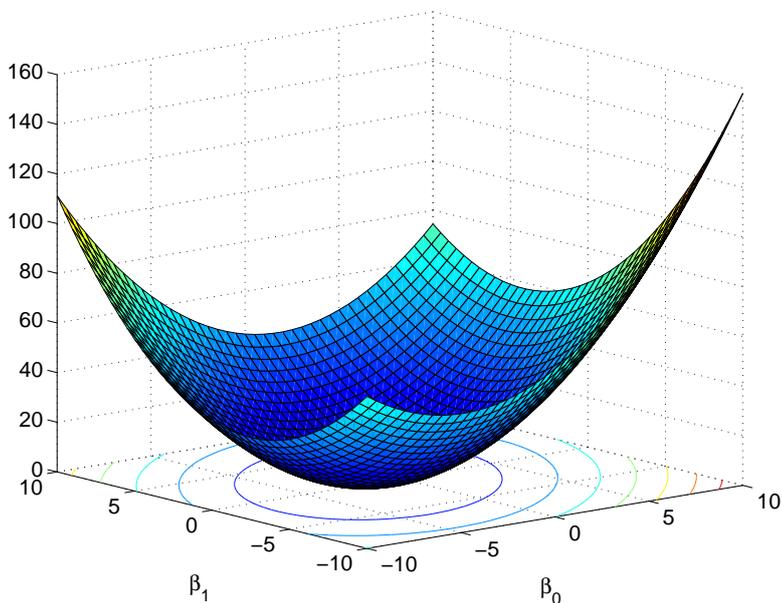
Are there any other issues?

Follow the gradient

A gradient (at a point) is the slope of the tangent (at that point). It points to the direction of largest increase of the function.

For 2-parameter model, MSE and MAE are shown below.

(I used $\mathbf{y}^T = [2, -1, 1.5]$ and $\mathbf{x}^T = [-1, 1, -1]$).



Batch gradient descent

To minimize the function, take a step in the (opposite) direction of the gradient

$$\boldsymbol{\beta}^{(k+1)} \leftarrow \boldsymbol{\beta}^{(k)} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial \boldsymbol{\beta}}$$

where $\alpha > 0$ is the [step-size](#) (or [learning rate](#)).

Gradient descent for 1-parameter model to minimize MSE:

$$\beta_0^{(k+1)} = (1 - \alpha)\beta_0^{(k)} + \alpha\bar{y}$$

Where $\bar{y} = \sum_n y_n / N$. When is this sequence guaranteed to converge?

Gradients for MSE

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \quad \tilde{\mathbf{X}} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix} \quad (1)$$

We define the error vector \mathbf{e} :

$$\mathbf{e} = \mathbf{y} - \tilde{\mathbf{X}}\boldsymbol{\beta} \quad (2)$$

and MSE as follows:

$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2N} \sum_{n=1}^N (y_n - \tilde{\mathbf{x}}_n^T \boldsymbol{\beta})^2 \quad (3)$$

$$= \frac{1}{2N} \mathbf{e}^T \mathbf{e} \quad (4)$$

then the gradient is given by,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = -\frac{1}{N} \tilde{\mathbf{X}}^T \mathbf{e} \quad (5)$$

What is the computational complexity?

Implementation Issues

Stopping criteria: When \mathbf{g} is (close to) zero, we are at (or close to) the optimum. If second-order derivative is positive, it is a minimum. See the section on [Optimality conditions](#).

Step-size selection: If α is too big, the method might diverge. If it is too small, convergence is slow. Convergence to a local minimum is guaranteed only when $\alpha < \alpha_{min}$ where α_{min} is a fixed constant that depends on the problem.

Line-search methods: We can set step-size automatically using a line-search method. More details on “backtracking” methods can be found in Chapter 1 of Bertsekas’ book on “nonlinear programming”.

Feature normalization: Gradient descent is very sensitive to ill-conditioning. Therefore, always normalize your feature. With unnormalized features, step-size selection is difficult since different “directions” might move at different “speed”.

Additional Notes

Pseudo-code for grid search

Pseudo-code for grid-search with MSE for 1-parameter model.

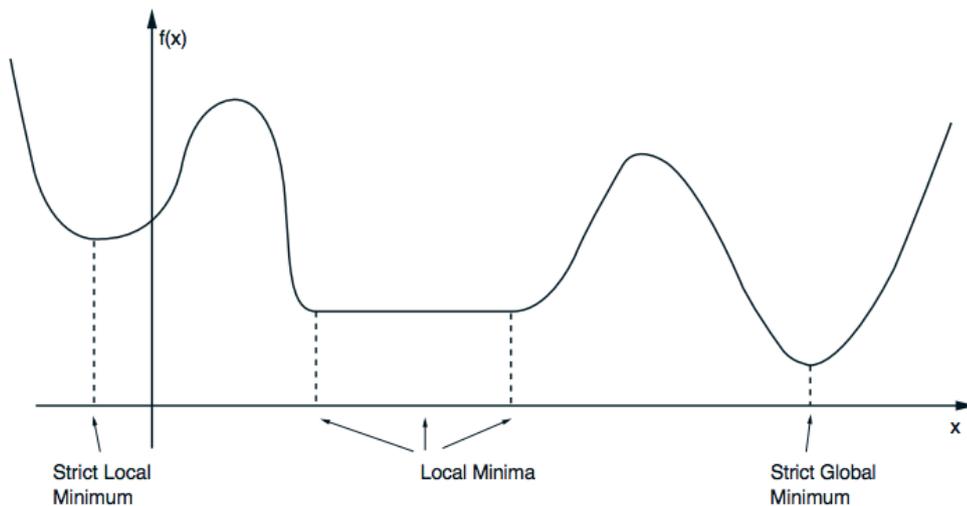
```
1 beta0 = -10:.1:10;
2 for i = 1:length(beta0)
3     err(i) = computeCost(y, X, beta0(i));
4 end
5 [val, idx] = min(err);
6 beta0_star = beta0(idx);
```

A function for computing MSE. We normalized MSE by $2N$ here. This keeps MSE normalized for different N (without affecting the minimum).

```
1 function computeCost(y, X, beta0)
2     e = y - beta0;
3     return e'*e / (2*N);
```

- Extend this code to a 2-parameters model.
- What is the computational complexity in terms of M (number of grid points), N (number of data examples) and D (number of dimensions) for grid search to minimize MSE for linear regression?
- Read more about grid search and other methods for “hyperparameter” setting: https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search.

Local and global minimum



The above figure is taken from Bertsekas, Nonlinear programming.

A vector β^* is a **local minimum** of \mathcal{L} if it is no worse than its neighbors; i.e. there exists an $\epsilon > 0$ such that,

$$\mathcal{L}(\beta^*) \leq \mathcal{L}(\beta), \quad \forall \beta \text{ with } \|\beta - \beta^*\| < \epsilon$$

A vector β^* is a **global minimum** of \mathcal{L} if it is no worse than all other vectors,

$$\mathcal{L}(\beta^*) \leq \mathcal{L}(\beta), \quad \forall \beta \in \mathbb{R}^{D+1}$$

These local or global minimum are said to be **strict** if the corresponding inequality is strict for $\beta \neq \beta^*$.

When working with a cost function, it is essential to make sure that a global minimum exist, i.e. that the cost function is lower bounded.

For more details to gain understanding about local and global minima, read the first 3 pages of Chapter 1 from Bertsekas' book on "Nonlinear programming".

Computational complexity

The **computation cost** is expressed using the **big-O** notation. Here is a definition taken from Wikipedia. Let f and g be two functions defined on some subset of the real numbers. We write $f(x) = O(g(x))$ as $x \rightarrow \infty$, if and only if there exists a positive real number c and a real number x_0 such that $|f(x)| \leq c|g(x)|$, $\forall x > x_0$.

Please read and learn more from this page in Wikipedia:

http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#Matrix_algebra.

- What is the computational complexity of the dot product $\mathbf{a}^T \mathbf{b}$ of two vectors \mathbf{a}, \mathbf{b} , both of length N ?
- What is the computational complexity of matrix multiplication?

Pseudo-code for gradient descent

Matlab implementation.

```
1 beta = zeros(D, 1);
2 for k = 1:maxIters
3     g = computeGradient(y, X, beta);
4     beta = beta - alpha * g;
5     if g'*g < 1e-5; break; end;
6 end
```

Compute gradient.

```
1 function computeGradient(y, X, beta)
2     % Fill this in
3     return g;
4 end
```

- What is the computational complexity (in terms of M , N and D) of grid search with MSE for linear regression?

Optimality conditions

The first-order *necessary* condition says that at *an* optimum the gradient is equal to zero.

$$\frac{\partial \mathcal{L}(\boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0 \quad (6)$$

The second-order *sufficient* condition ensures that the optimum is a minimum (not a maximum or saddle-point) using the [Hessian](#) matrix.

$$\mathbf{H}(\boldsymbol{\beta}^*) := \frac{\partial^2 \mathcal{L}(\boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} \text{ is positive definite.} \quad (7)$$

The Hessian is also related to the convexity of a function: a twice-differentiable function is convex if and only if the Hessian is positive definite.

Stochastic gradient descent

When N is large, choose a random pair (\mathbf{x}_i, y_i) in the training set and take a step.

$$\boldsymbol{\beta}^{(k+1)} \leftarrow \boldsymbol{\beta}^{(k)} + \alpha^{(k)} \frac{\partial \mathcal{L}_n(\boldsymbol{\beta}^k)}{\partial \boldsymbol{\beta}}$$

For convergence, $\alpha^k \rightarrow 0$ “appropriately”. One such condition called Robbins-Monroe condition suggests to take α^k such that:

$$\sum_{k=1}^{\infty} \alpha^{(k)} = \infty, \quad \sum_{k=1}^{\infty} (\alpha^{(k)})^2 < \infty \quad (8)$$

One way to obtain such sequence is $\alpha^{(k)} = 1/(1+k)^r$ where $r \in (0.5, 1)$.

Read Section 8.5 of Kevin Murphy’s book. There are many variants of this method too. See more details and references in the Wikipedia page https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

To do

1. Revise computational complexity (also see the Wikipedia link in Page 6 of lecture notes).
2. Derive computational complexity of grid-search and gradient descent.
3. Derive gradients for MSE and MAE cost functions.
4. Derive convergence of gradient descent for 1 parameter model.
5. Implement gradient descent and gain experience in setting the step-size.
6. Learn to assess convergence of gradient descent.
7. Revise linear algebra to understand positive-definite matrices.
8. Implement stochastic gradient descent and gain experience in setting the step-size.