Gradient Descent

Mohammad Emtiyaz Khan
EPFL

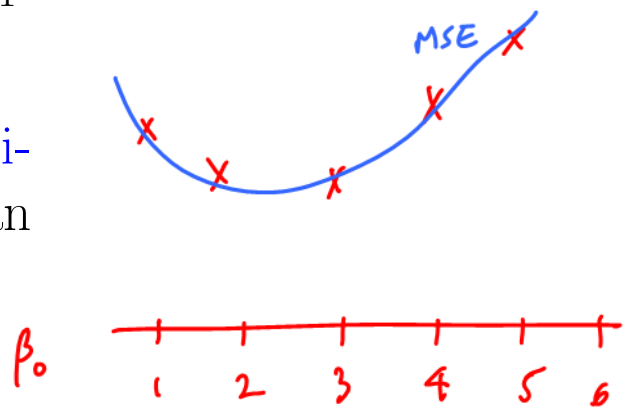
Sep 22, 2015

Learning/estimation/fitting

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

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

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.

grid points

$O(NDM^D)$

$\sum_n (y_n - \beta_0 - \beta_1 x_{n1})^2$

$[\beta_0\ \beta_1\ \beta_2 \cdots \beta_D] \begin{bmatrix} 1 \\ x_{n1} \\ x_{n2} \\ \vdots \\ x_{nD} \end{bmatrix}$

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

READ THE PSEUDO CODE

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.

Read about computational complexity (see last section)

Are there any other issues? Local minimum might make it difficult to search the minimum.

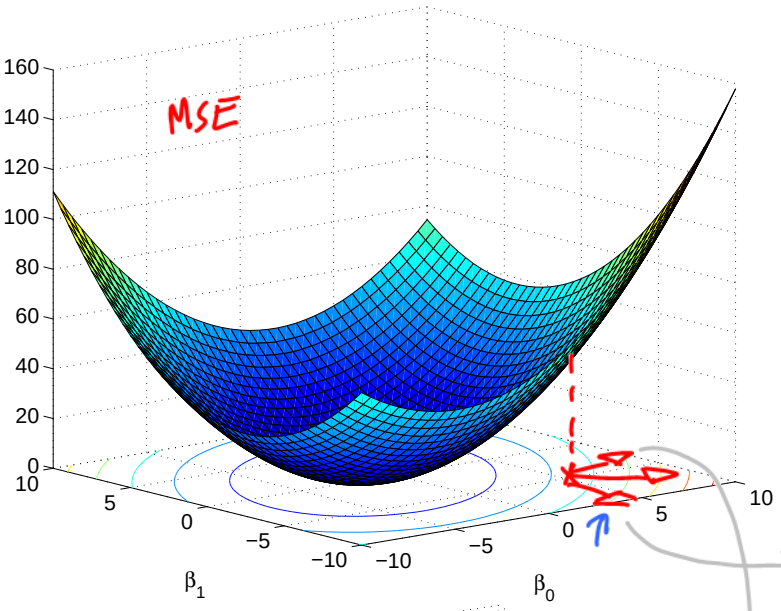
(see last section.)

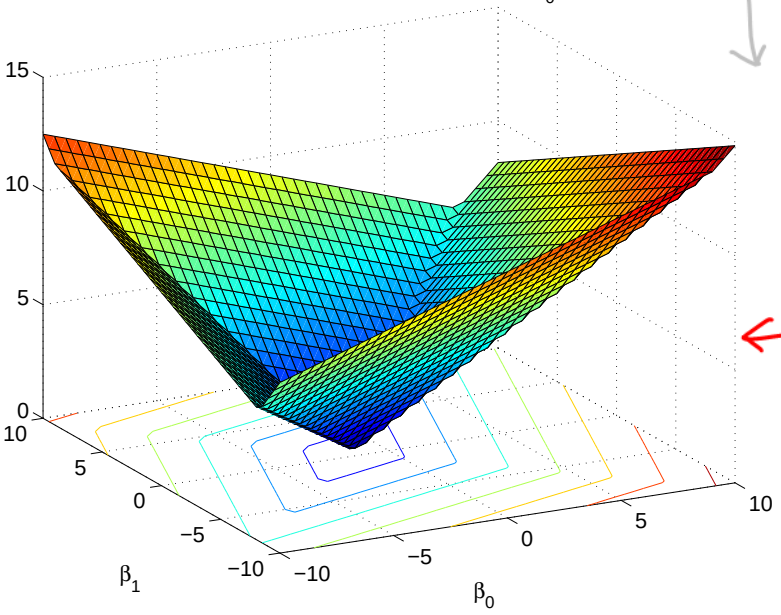
Follow the gradient

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.

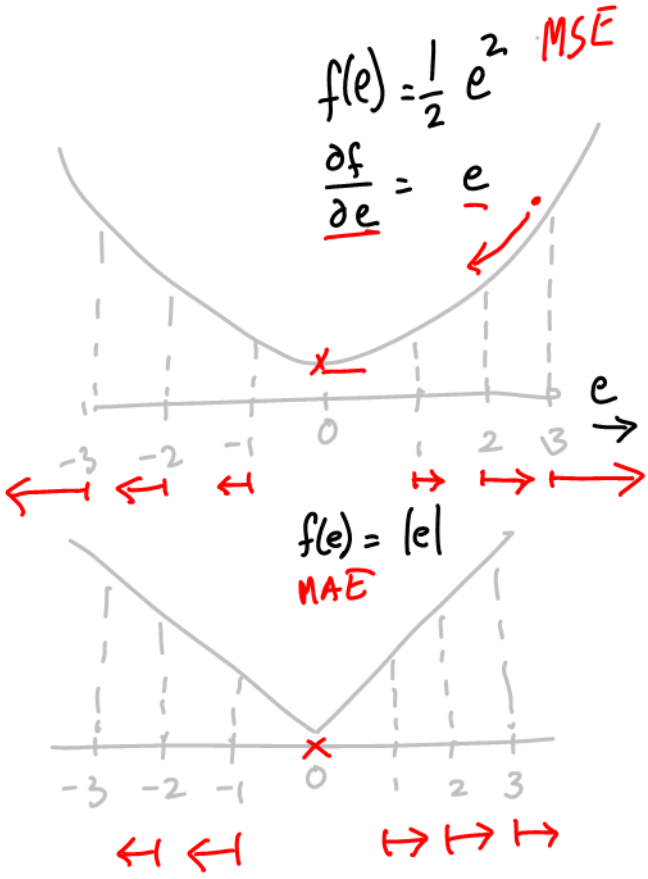
For 2-parameter model, MSE and MAE are shown below.

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





← MAE



$$\mathcal{L}(\beta_0, \beta_1) = \frac{1}{2N} \sum_n (y_n - \beta_0 - \beta_1 x_{n1})^2$$

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = -\frac{1}{N} \sum_{n=1}^{N} \underbrace{(y_n - \beta_0 - \beta_1 x_{n1})}_{e_n(\beta_0, \beta_1)} 1$$

$$\frac{\partial \mathcal{L}}{\partial \beta_1} = -\frac{1}{N} \sum_n \underbrace{(y_n - \beta_0 - \beta_1 x_{n1})}_{e_n(\beta_0, \beta_1)} x_{n1}$$

Batch gradient descent

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

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

Notation

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?

THINK-PAIR-SHARE

$$\mathcal{L}(\beta_0) = \frac{1}{2N}\sum_{n=1}^{N}(y_n - \beta_0)^2$$

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = -\frac{1}{N}\sum_n (y_n - \beta_0)$$

$$\frac{\partial \mathcal{L}(\beta_0^{(k)})}{\partial \beta_0} = -\frac{1}{N}\sum_n (y_n - \beta_0^{(k)})$$

$$\beta_0^{(k+1)} = \beta_0^{(k)} + \alpha\frac{1}{N}\sum(y_n - \beta_0^{(k)})$$

$$= (1-\alpha)\beta_0^{(k)} + \alpha\underbrace{\frac{1}{N}\sum_{n=1}^{N} y_n}_{\bar{y}}$$

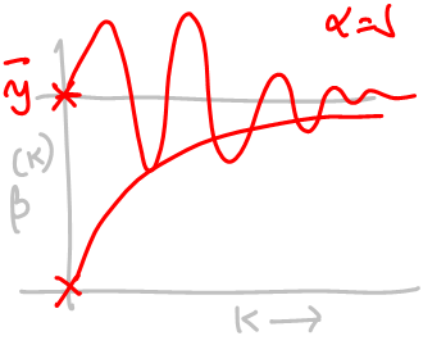
Say, $\beta_0^{(0)} = 0$,

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

$$\beta_0^{(2)} = (1-\alpha)\beta_0^{(1)} + \alpha\bar{y} = [(1-\alpha)+1]\alpha\bar{y}$$

$$\beta_0^{(3)} = (1-\alpha)\beta_0^{(2)} + \alpha\bar{y} = [(1-\alpha)^2 + (1-\alpha) + 1]\alpha\bar{y}$$

$$\beta_0^{(k)} = [(1-\alpha)^{k-1} + (1-\alpha)^{k-2} + \cdots + 1]\alpha\bar{y}$$



$$= \frac{1-(1-\alpha)^k}{1-(1-\alpha)}\alpha\bar{y} = \underbrace{[1-(1-\alpha)^k]\bar{y}}$$

$\alpha < \alpha_{max}$

Gradients for MSE

(Revise Matrix differentiation, use "Matrix Cookbook" as reference)

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

$\widetilde{\mathbf{x}}_1^T$, $\widetilde{\mathbf{x}}_2^T$, $\widetilde{\mathbf{x}}_N^T$

$\widetilde{\mathbf{X}} \in \mathbb{R}^{N \times (D+1)}$

We define the error vector $\mathbf{e}$:

$$\mathbf{e} \triangleq \mathbf{y} - \widetilde{\mathbf{X}}\boldsymbol{\beta} \quad (2)$$

$$\underline{e} \triangleq \underline{y} - \widetilde{X}\underline{\beta} = \begin{bmatrix} y_1 - \widetilde{\mathbf{x}}_1^T\underline{\beta} \\ y_2 - \widetilde{\mathbf{x}}_2^T\underline{\beta} \\ \vdots \\ y_N - \widetilde{\mathbf{x}}_N^T\underline{\beta} \end{bmatrix} \begin{matrix} \leftarrow e_1 \\ \leftarrow e_2 \\ \\ \leftarrow e_N \end{matrix}$$

and MSE as follows:

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

$(y_n - \widetilde{\mathbf{x}}_n^T\boldsymbol{\beta})^2 = e_n^2$

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

$$[e_1 \; e_2 \; \dots \; e_N] \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} = \sum_n e_n^2$$

then the gradient is given by,

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

What is the computational complexity?

This is a heuristic to derive the gradient

$$\frac{\partial \mathcal{L}}{\partial \underline{e}} = \frac{1}{2N} 2\underline{e} = \frac{1}{N}\underline{e}$$

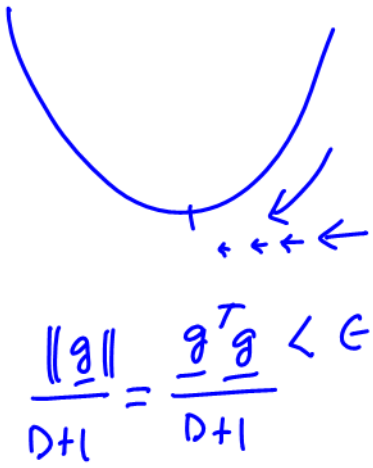
$N \times 1$ $\quad$ $N \times 1$

$$\frac{\partial \mathcal{L}}{\partial \underline{\beta}} = \frac{\partial \mathcal{L}}{\partial \underline{e}} \frac{\partial \underline{e}}{\partial \underline{\beta}} = \frac{-1}{N}\underline{e}X$$

$D \times 1$ $\quad$ $D \times N$ $\quad$ $N \times 1$

$$= -\frac{1}{N}X^T e$$

Implementation Issues

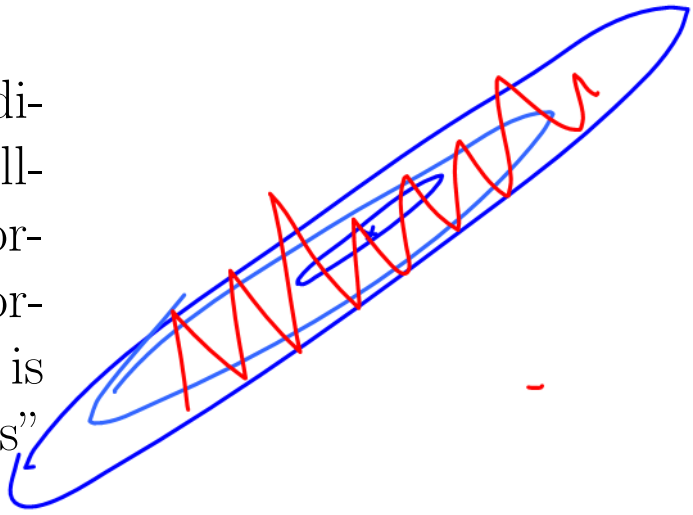


Stopping criteria: When $\mathbf{g}$ is (close to) zero, we are at (or close to) the optimum. If second-order derivative is positive, it is a minimum. See the section on Optimality conditions.

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 selecion 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.

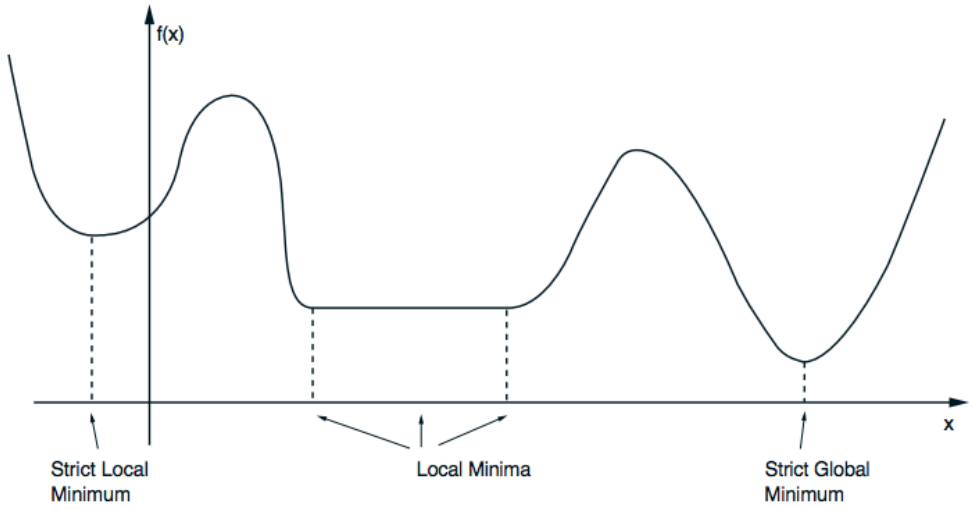
```
beta0 = -10:.1:10;
for i = 1:length(beta0)
  err(i) = computeCost(y, X, beta0(i));
end
[val, idx] = min(err);
beta0_star = beta0(idx);
```

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

```
function computeCost(y, X, beta0)
  e = y - beta0;
  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 $\boldsymbol{\beta}^*$ is a local minimum of $\mathcal{L}$ if it is no worse than its neighbors; i.e. there exists an $\epsilon > 0$ such that,

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

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

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

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

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 $x \to \infty$, if and only if there exists a positive real number c and a real number x_0 such that $|f(x)| \leq c|g(x)|, \quad \forall x > x_0$.

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.

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

Compute gradient.

```
function computeGradient(y,X,beta)
  % Fill this in
  return g;
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.

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

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

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

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, $\alpha^k \to 0$ "appropriately". One such condition called Robbins-Monroe condition suggests to take α^k such that:

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

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

Read Section 9.5 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.

Update: Some say it's 8.5 depending on the edition!

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.