Machine Learning

Supervised learning 3

S. Nõmm

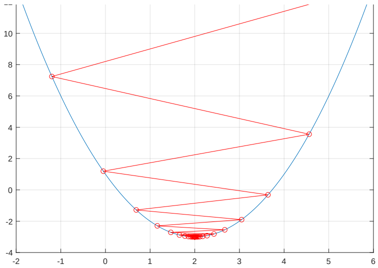
[1]Department of Software Science, Tallinn University of Technology

05.03.2024

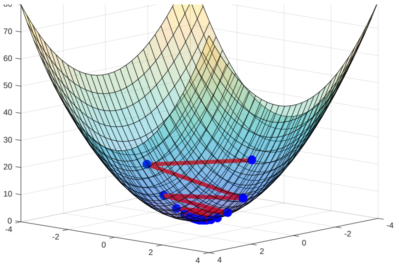
Gradient descent

- Is the first order optimization algorithm.
- Requires to compute first order derivatives.
- Basic idea: If multi-variable function $f(x)$ is defined and differentiable in a neighbourhood of point x_0, then fastest decrease of $f(x)$ is in the direction of the negative gradient $\nabla f(x)$ from the point x_0.
- Formally: Let $x_{k+1} = x_k - \eta_k \nabla f(x_k)$ then $f(x_0) \geq f(x_1) \geq f(x_2) \geq \dots$. Where η_k is the learning rate. In this form it is referred ad steepest descent.
- This method allows to find local minima of the function.
- In the case of convex function local minima are also global.
- Observe the fact that here we talk about finding minimum of a function in general. (Notation is chosen accordingly). In the case of objective function f is the objective function and its argument are the parameters θ.

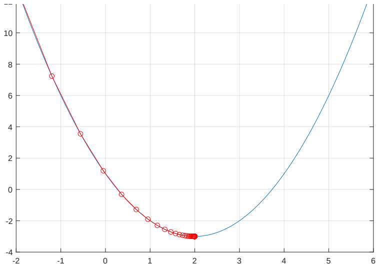
Examples



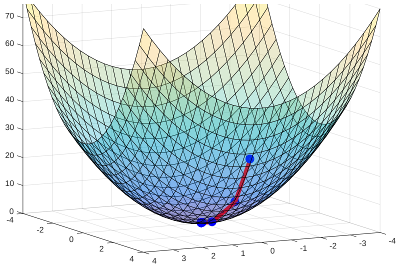
10
8
6
4
2
0
-2
-4
-2
-1
0
1
2
3
4
5
6



70
60
50
40
30
20
10
0
-4
-2
0
2
4
4
2
0
-2
-4



10
8
6
4
2
0
-2
-4
-2
-1
0
1
2
3
4
5
6



70
60
50
40
30
20
10
0
-4
-2
0
2
4
4
3
2
1
0
-1
-2
-3
-4

Learning rate (step size)

- "Zig-zagging" effect observed in first row (previous slide).
- Choose η to be small enough. Lead slow convergence. (May not reach minimum at all.)
- Line search: Find η to minimize

$$\phi(\eta) = f(x_k + \eta \mathrm{d}_k)$$

  where d is the descent direction.
- Heavy ball method. Add a *momentum term*:

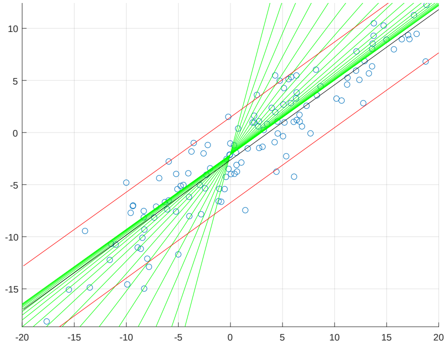
$$x_{k+1} = x_k - \eta_k \nabla f(x_k) + \mu_k (x_k - x_{k-1})$$

  where $0 \leq \mu_k \leq 1$
- Another technique:

$$\eta_k = \frac{(x_k - x_{k-1})^T (\nabla f(x_k) - \nabla f(x_{k-1}))}{||\nabla f(x_k) - \nabla f(x_{k-1})||^2}$$

What actually happening with your model

Please be reminded that on each iteration you are updating the weights of your model.



As a result of each iteration one have a model (values of the parameters) (not accurate) of the modelled process.

Newton's method

- Takes into account curvature of the space. Requires to compute Hessian.
- Update is given by:

$$x_{k+1} = x_k - \eta_k \mathrm{H}_k^{-1} g_k$$

  where $g_k = \nabla f(x_k)$ and $\mathrm{H}_k = \nabla^2 f(x_k)$.
- This method requires that H_k is positive define. (H_k is positive define if $f(x)$ is strictly convex).
- Levenberg Marquardt algorithm may be seen as hybrid of Newton's and steepest descent. Whenever possible it is Newton but when hessian is not positive define it applies ordinary steps of steepest descent.

Logistic regression

- Remind that linear regression may be written in the following form:

$$p(y|x,\theta) = \mathcal{N}(y|\mu(x), \sigma^2(x))$$

- This may be generalized to the binary setting as follows:

$$p(y|x,\theta) = \mathrm{Ber}(y|\mathrm{sigm}(\theta^T x))$$

  where $\mathrm{sigm}(z) = (1+e^{-z})^{-1}$. Will be referred as *logistic regression*.
- Assume that $z = b_1 x + b_0$ then

$$\mathrm{sigm}(z(x)) = \frac{1}{1+e^{-(b_1 x + b_0)}}$$

Logistic regression

- Derivative of logistic function:

$$g(z) = \frac{1}{1+e^{-z}} \Rightarrow g'(z) = g(z)(1-g(z)).$$

- Probabilistic interpretation:

$$\begin{aligned} P(y=1|x;\theta) &= g(\theta^T x) \\ P(y=0|x;\theta) &= 1 - g(\theta^T x) \end{aligned}$$

- The term $\theta^T x$ referred as log-odds. Odds refers the ratio of probability occurring to the probability not occurring.

Fitting logistic model

- Let $h_\theta(x)g(\theta^T x) = (1 - e^{-\theta^T x})^{-1}$
- Fitting is usually done by maximum likelihood. (Log likelihood is preferred.)

$$\ell(\theta) = \log \prod_{i=1}^{m} h_\theta(x_i)^{y_i}(1 - h_\theta(x_i)^{1-y_i})$$

- Solving the last one lead update rule.

$$\theta_j^{(k+1)} = \theta_j^k + \gamma \sum_{i=1}^{m} (y_i - h_\theta(x_i))x_{i,j}$$

ℓ_2 regularization

- The case of regularization is not general therefore switch to the notations of minimization of cost function.
- Ridge regression is preferred in the case of linear regression.
- In the case of logistic regression

$$\begin{aligned} f'(\theta) &= \mathrm{NLL}(\theta) + \lambda\theta^t\theta \\ g'(\theta) &= g(\theta) + \lambda\theta \\ H'(\theta) &= H(\theta) + \lambda I \end{aligned}$$

Online learning

- Suppose that at each step a sample z_k is presented and "learner" is expected to respond by giving parameter estimate θ_k.
- In the case of on-line learning objective is the *regret*.

$$\mathcal{P}_k = \frac{1}{k}\sum_{t=1}^{k} f(\theta_t, z_t) - \min_{\theta^* \in \Theta} \frac{1}{k}\sum_{t=1}^{k} f(\theta^*, z_t)$$

- Loss function in this case $f(\theta, z) = -\theta^T z$ and *regret* is how much batter/worst one did by adopting different strategy.
- Online gradient descent:

$$\begin{aligned}
\theta_{k+1} &= \mathrm{proj}_{\Theta}(\theta_k - \eta_k g_k) \\
\mathrm{proj}_{\mathcal{V}}(v) &= \arg\min_{\mathrm{w} \in \mathrm{V}} ||\mathrm{w} - \mathrm{v}||_2 \\
g_k &= \nabla f(\theta_k, z_k)
\end{aligned}$$

Stochastic optimization and risk minimization

- Contrary to minimizing regret one may minimize expected loss. $f(\theta) = \mathbb{E}[f(\theta, z)]$.
- One have to optimize functions where some variables in the objective are random. This is called *stochastic optimization*.
- Stochastic gradient descent:

$$\bar{\theta} = \frac{1}{k}\sum_{t=1}^{k}\theta_t$$

whereas recursive implementation is as follows:

$$\bar{\theta}_k = \bar{\theta}_{k-1} - \frac{1}{k}(\bar{\theta}_{k-1} - \theta_k)$$

The perceptron

- Fitting logistic regression model in an online manner.
- The weight update rule is given by:

$$\theta_k = \theta_{k-1} - \eta_k g_i = \theta_{k-1} - \eta_k(\mu_i - y_i)x_i$$

  where $\mu_i = p(y_i = 1|x_i, \theta_k) = \mathbb{E}[y_i|x_i, \theta_k]$
- Consider an approximation of this algorithm, denote most probable class label as

$$\hat{y} = \arg\max p(t|x_i, \theta)$$

- The update rule may be simplified as $\theta_k = \theta_{k-1} + \eta_k y_i x_i$.
- The gradient expression $g_i = x_i(\theta_k^T x_i - y_i)$ becomes approximate:

$$g_i \approx (\hat{y}_i - y_i)x_i$$

- Assume that $y \in \{-1, 1\}$ not just 0 or 1 then prediction becomes

$$\hat{y}_i = \operatorname{sign}(\theta^T, x_i)$$

- Final update rule is given by

$$\theta_k = \theta_{k-1} + \eta_k y_i x_i$$

The perceptron

- If the classes are linearly separable the algorithm will converge.
- Preferable for two -class problems.

More general view

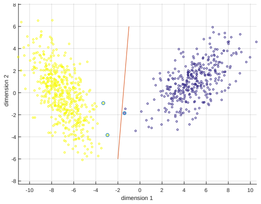
- Linear regression

$$\hat{y} = a_1 x_1 + \dots a_n x_n + b = x^T a + b$$

- Logistic regression (case of two classes)

$$C(x) = \text{sign}[x^T a + b]$$

One may think about trend line (it is not any more a trendline) as of hypersurface (decision boundary) separating space into subspaces corresponding to each class.



Separability

Linear separability

Two classes are said to be linearly separable in $\mathbb{R}^n$ -if there exists a hyperplane dividing the space into two subspaces such that all the elements of the first class belong to one subspace and the elements of the second class belong to the other subspace.
Or
-if there exist n - dimensional vector a and scalar b such that for the elements of one class $x^T a > b$ and for the elements of the second class $x^T a < b$

Separability

- If two classes are linearly separable it is possible to construct two hyperplanes, parallel to the ”separating” hyperplane, such that first hyperplane would contain at least one point of the first class and second hyperplane will contain at least one point of the second class.
- The training data points belonging to these hyperplanes are referred as *support vectors* and the distance between the hyperplanes is referred as *margin*.
- In order to determine maximum margin hyperplane nonlinear programming optimization is required. First margin is expressed as the function of the coefficients of separating hyperplane. Second optimization problem is solved.

Maximum margin hyperplane

- For the hyperplane $x^T a + b$ vector $a = (a_1, \ldots, a_n)$ is n - dimensional vector representing the normal direction to the hyperplane.
- Then the distance (margin) from the separating hyperplane to the hyperplanes containing points of each class (see previous slide) would be $M = ||a||^{-1}$.
- The optimization problem may be stated in terms of finding vector a that would maximize margin

Self practice

- Program your own implementation gradient descent.
- Program your own implementation Newton's method.