UiT Norges arktiske universitet

DTE-2501 AI Methods and Applications

Linear methods of classification and regression

Lecture 1/2 – Stochastic gradient descent

Tatiana Kravetc
Førsteamanuensis

Office: D2240

Email: tatiana.kravetc@uit.no

Overview

I Linear model

II Numerical minimization

a) Gradient descent

b) Stochastic gradient

c) Extensions and their comparison

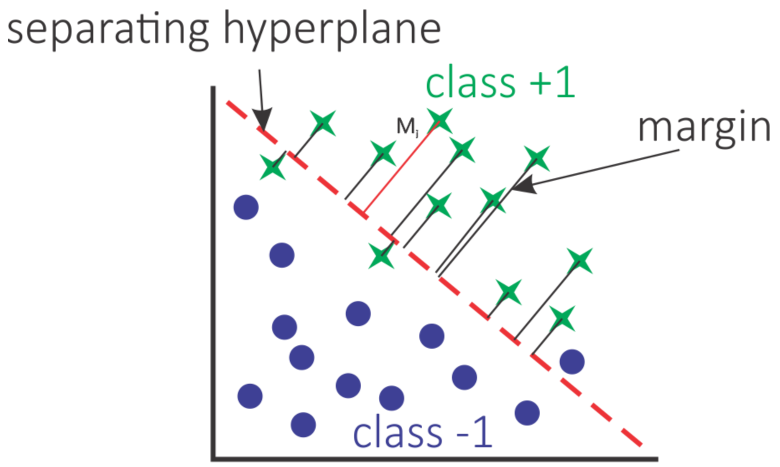
I Linear model

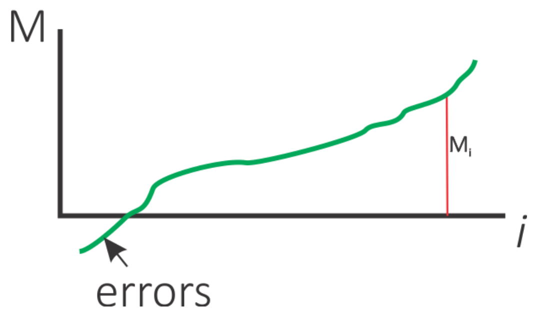
Regression	Classification
• Training sample $X^l = (x_i, y_i)_{i=1}^l, x_i \in \mathbb{R}^n, y_i \in \mathbb{R}$ • Linear model. $\langle\cdot,\cdot\rangle$ is a scalar product $a(x,w) = \langle x,w\rangle = \sum_{j=1}^n w_j f_j(x),\ w \in \mathbb{R}^n$ • Quadratic *loss function* $\varepsilon(a,y) = (a(x,w) - y(x))^2$ • Training phase. Learning method is a *least squares method*. Q is an *empirical risk* $Q(w) = \sum_{i=1}^{l} (a(x_i,w) - y_i)^2 \to \min_w$ • Testing phase. Test sample $X^k = (\tilde{x}_i, \tilde{y}_i)_{i=1}^k$ $\tilde{Q}(w) = \frac{1}{k}\sum_{i=1}^{k} (a(\tilde{x}_i,w) - \tilde{y}_i)^2$	• Training sample $X^l = (x_i, y_i)_{i=1}^l, x_i \in \mathbb{R}^n, y_i \in \{-1,+1\}$ • Linear model $a(x,w) = \mathrm{sign}\langle x,w\rangle = \mathrm{sign}\sum_{j=1}^n w_j f_j(x)$ • Binary loss function, or its *approximation* $\varepsilon(a,y) = [a(x,w)y(x) < 0] = [\langle x,w\rangle y < 0] \le \varepsilon(\langle x,w\rangle y)$ • Training phase. Learning method is a minimization of the empirical risk $Q(w) = \sum_{i=1}^{l} [a(x_i,w)y_i < 0] = \sum_{i=1}^{l} \varepsilon(\langle x_i,w\rangle y_i) \to \min_w$ • Testing phase. Test sample $X^k = (\tilde{x}_i, \tilde{y}_i)_{i=1}^k$ $\tilde{Q}(w) = \frac{1}{k}\sum_{i=1}^{k} [\langle \tilde{x}_i,w\rangle \tilde{y}_i < 0]$

Margin

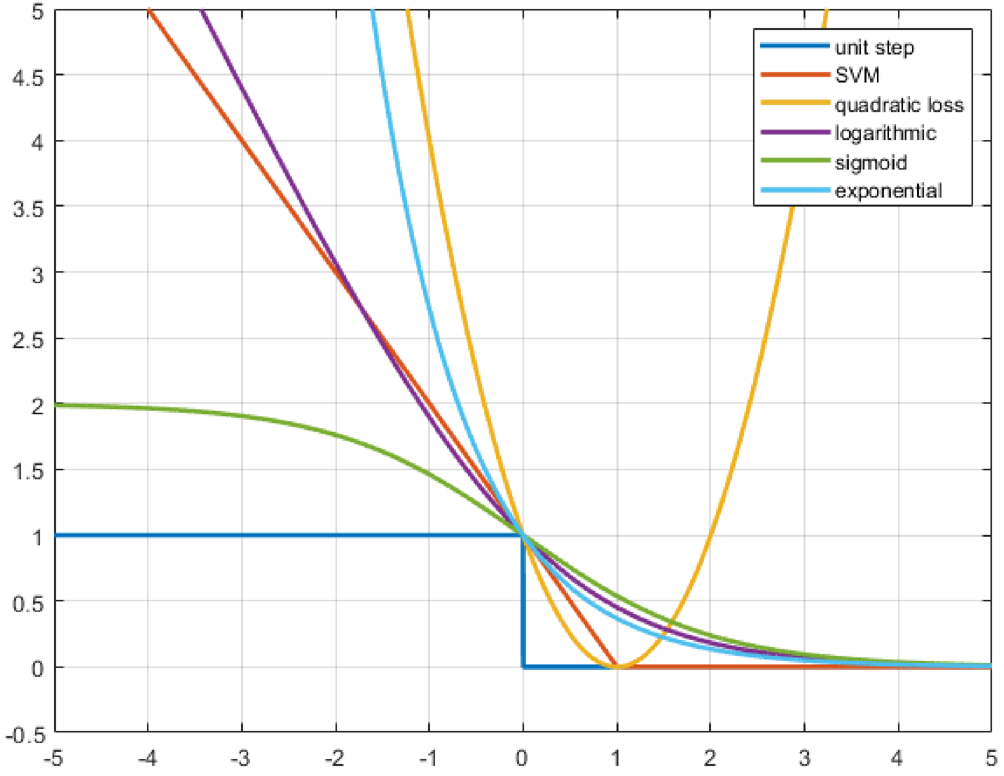
Classifier $a(x, w) = \mathrm{sign}\langle x, w\rangle$

- $\langle x, w\rangle = 0$ is a separating hyperplane equation
- $M_i = \langle x_i, w\rangle y_i$ is a margin of the object x_i
- $M_i < 0$ means an error of the algorithm $a(x, w)$ on the object x_i





Continuous approximations of the threshold function



- $[M < 0]$ – the unit step (zero-one loss)
- $(1 - M)_+$ – piecewise linear (SVM)
- $(1 - M)^2$ – quadratic loss
- $\log_2(1 + e^{-M})$ – logarithmic
- $2(1 + e^{M})^{-1}$ – sigmoid
- e^{-M} – exponential

II Numerical minimization

Gradient descent

Empirical risk minimization

$$Q(w) = \sum_{i=1}^{l} \varepsilon_i(w) \to \min_w$$

Numerical minimization using a *gradient descent method:*

$w^{(0)}$ is an initial guess

$$w^{(t+1)} \coloneqq w^{(t)} - h \cdot \nabla Q(w^{(t)}), \quad \nabla Q(w) = \left(\frac{\partial Q(w)}{\partial w_j}\right)_{j=0}^{n}$$

where h is a gradient step, which is also called a *learning rate.*

$$w^{(t+1)} \coloneqq w^{(t)} - h \sum_{i=1}^{l} \nabla \varepsilon_i\big(w^{(t)}\big)$$

Stochastic gradient

Input: dataset X^l, learning rate h, parameter λ
Output: weights w

Initialization
Set all the weights $w_j, j = 0, \ldots, n$ to small random numbers
Evaluate the objective function $Q = \frac{1}{l}\sum_{i=1}^{l} \varepsilon_i(w)$

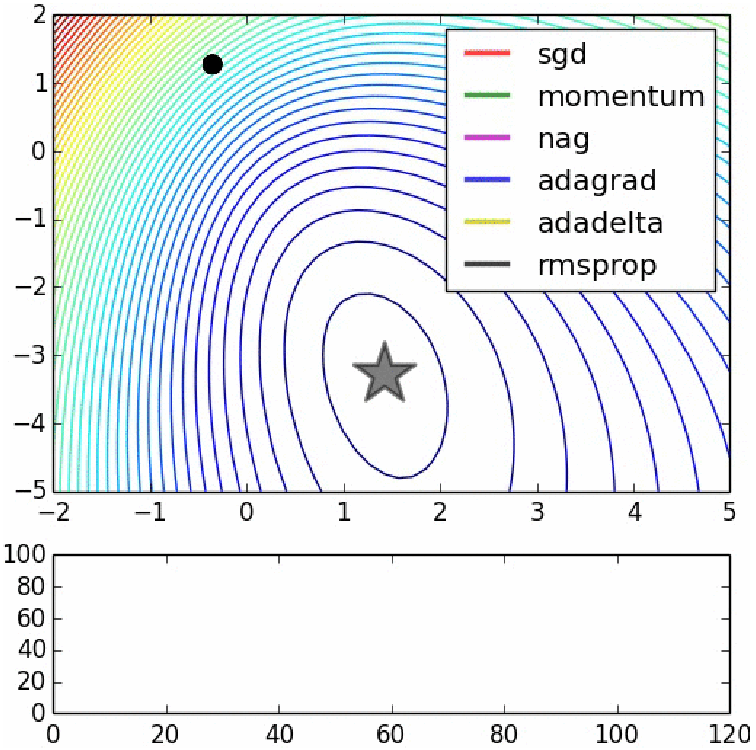
do
 pick randomly x_i from X^l
 compute the loss function $\varepsilon_i(w)$
 perform the gradient step $w \coloneqq w - h\nabla\varepsilon_i(w)$
 update the objective function $Q \coloneqq \lambda\varepsilon_i + (1-\lambda)Q$
until Q and/or w converges

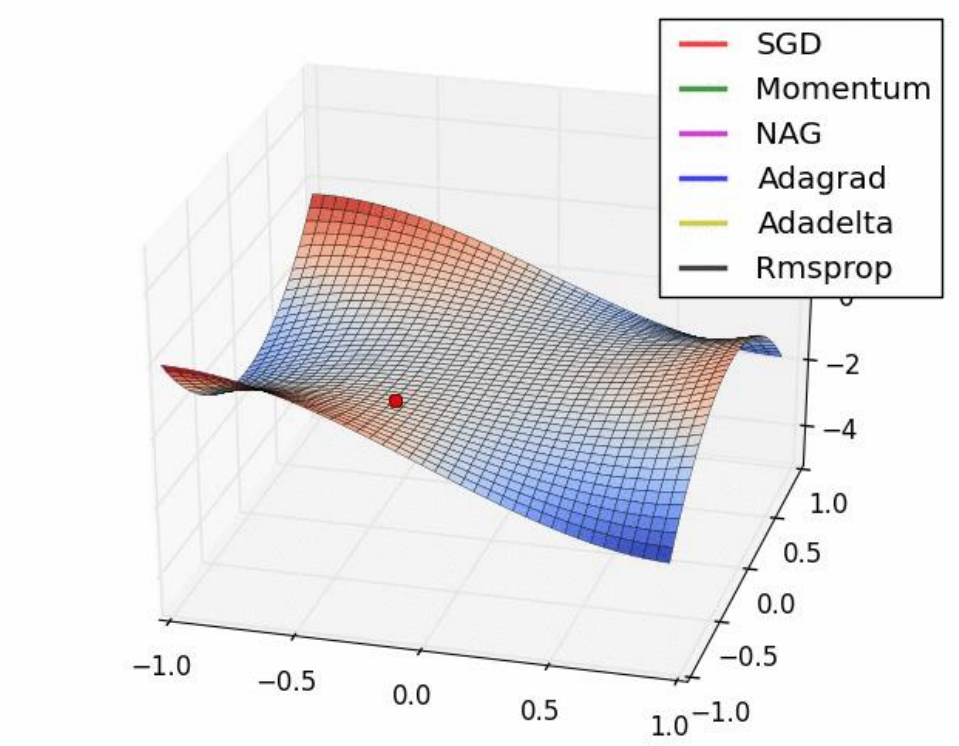
Robbins, H., Monro, S. (1951). A Stochastic Approximation Method. The Annals of Mathematical Statistics, 22(3), 400–407

Extensions

- Stochastic average gradient (SAG)
  The algorithm records an average of its parameter vector over time
- Momentum
  The algorithm updates the weights as a linear combination of the gradient and the previous update
- RMSProp (running mean square propagation)
  The learning rate is adapted for each of the parameters
- AdaGrad (adaptive gradient)
  The algorithm increases the learning rate for sparser parameters and decreases for ones that are less sparse
- etc…

Comparison of optimization algorithms





Alec Radford's animation for optimization algorithms:

http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

III Practical example

Objective function $Q = x \sin x$ on $[-8,8]$

Derivative of the objective function $Q' = \sin x + x \cos x$

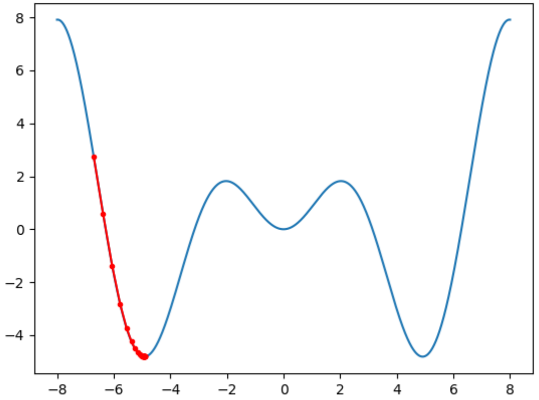
```python
# objective function
def obj(x):
    return x * sin(x)


# derivative of objective function
def derv(x):
    return sin(x) + x * cos(x)


# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size):
    solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
    for i in range(n_iter):
        gradient = derivative(solution)
        solution = solution - step_size * gradient
    solution_eval = objective(solution)
    return [solution, solution_eval]


bound = asarray([[-8.0, 8.0]])
n = 30
h = 0.1
sol, sol_eval = gradient_descent(obj, derv, bound, n, h)
```

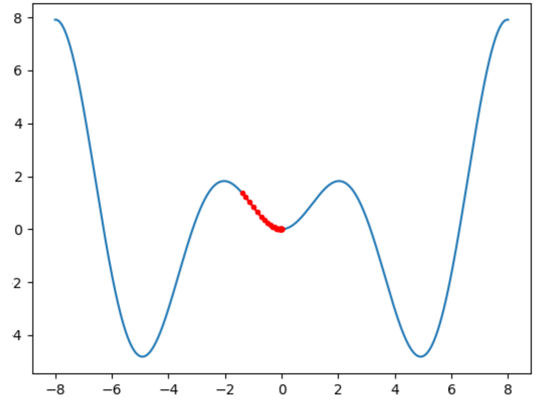
random initialization

$x := x - hQ'(x)$

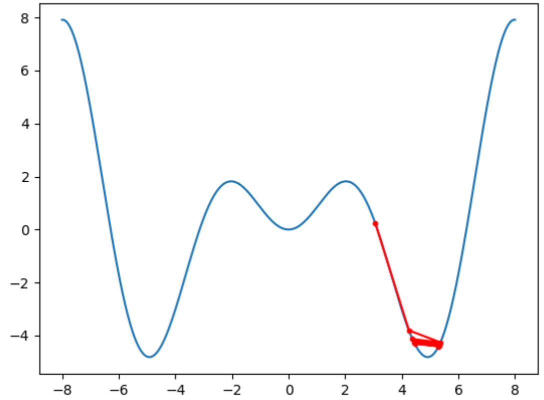
Result:

Local minima depends on starting point





Not optimal step size ($h = 0.4$)



https://machinelearningmastery.com/gradient-descent-optimization-from-scratch/