Gradient Descent

Wenxuan Wang

January 2021

1 Problem Formulation

Gradient descent is a common method to solve unconstrained optimization problems. It has a wide application in machine learning and statistics. Convex optimization is the most important problem in optimization methods. For machine learning, if the problem is proved to be a convex optimization problem, it means that the problem can be solved better.

1.1 Convex Function

1.1.1 Definition

We firstly let X be a convex subset of a real vector space and set a function $f \to \mathbb{R}$. For all $\theta \in [0,1]$ and all $x, y \in X$, $f(\theta x+(1-\theta)y) \leq \theta f(x)+(1-\theta)f(y)$, the function f can be called convex. For $\theta \in [0,1]$ and all $x, y \in X$ such that $x \neq y$, $f(\theta x+(1-\theta)y) \leq \theta f(x)+(1-\theta)f(y)$, the function f is strictly convex.

1.1.2 Properties

For the first-order judgement of convex function, we have $f(y) \geq f(x)+\triangledown f(x)^T(y-x)$ for all x, y in the interval. Geometrically, this uni-variate function is convex on an interval if and only if its graph lies above all of its tangents. For the second derivation, we have $f''(x) \geq 0$. If the second derivation is positive at all the points, the function is strictly convex. In the gradient descent problem, we have the extreme value where the derivation can be zero.

1.2 Convex Optimization

1.2.1 Definition

If the feasible region of an optimization problem is a convex set, and the objective function is a convex function, then the problem is a convex optimization problem. If for all $\theta \in [0,1]$ and all x, y in its domain, we have a function f mapping some subset of $\mathbb{R}^n$ into $\mathbb{R} \cup \{+\infty\}$ and the function meets the condition

$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$, the function f is said to be convex. Then, the convex optimization problem can be written as

$$minf(x)$$

$$x \in C$$

where x is the optimized variable; f is the convex objective function; C is the feasible region of the optimized variable. It is a convex set. The standard form of convex optimization problem can be written as

$$minf(x)$$

$$g_i(x) \leq 0, i = 1, ..., m$$

$$h_i(x) = 0, i = 1, ..., p$$

where $x \in \mathbb{R}^n$ is the optimization variable, the function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ is convex, $g_i : \mathbb{R}^n \to \mathbb{R}, i = 1, ..., m$, are convex and $h_i : \mathbb{R}^n \to \mathbb{R}, i = 1, ..., p$, are affine.

1.2.2 Properties

From the definition, we will have the thought to solve convex optimization problems, that is, to prove the objective function is convex. If the objective function is strictly convex, the problem will have at most one optimal point. Besides, for the optimal point, every local minimum is a global one.

2 Methodology

2.1 Principles

If the univariate function $f(x)$ is n-th order differential, its Taylor expansion is

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)(\Delta x)^2 + ... + \frac{1}{n!}f^(n)(x)(\Delta x)^n$$

The Taylor expansion of multivariate function is

$$f(X + \Delta X) = f(X) + (\nabla f(X))^T \Delta x + o(\Delta X)$$

In this equation, we have ignored second and even higher terms. Then, we can get

$$f(X + \Delta X) - f(X) \approx (\nabla f(X))^T \Delta X$$

For this formula, X is a multivariate, so it has many directions. It is unlike a univariate which only has a left and a right direction. To get a minimum value, we need to have $f(x + \Delta X) < f(X)$, so we should ensure $(\nabla f(X))^T \Delta X < 0$. In this inequality, its left term has $(\Delta f(X))^T \Delta X = \| \nabla f(X) \| \| \Delta X \| cos\theta$ where $\| \cdot \|$ is the modulus of vector, and θ is the angle between vectors. The modulus of vector must be greater than zero, so we only need to have $cos\theta \leq 0$. As $cos\theta = [-1, 1]$, when $\theta = \pi$, we can get the minimum value -1. Then $f(X + \Delta X)$ will have a maximum drop value along the gradient of $f(x)$. This drop value is $\parallel \nabla f(X) \parallel\parallel \Delta X \parallel$. We assume $\Delta X = -\alpha\nabla f(X)$, where α is the step size. It is an integer close to zero. Then we have $(\nabla f(X))^T\Delta X = -\alpha(\nabla f(X))^T(\nabla f(X)) \leq 0$. For the variable x, we have the iteration formula $X_{k+1} = X_k - \alpha\nabla f(X_k)$. Therefore, if there is no point where the gradient is zero, the function $f(x)$ will decrease in the direction of the negative descent. The end condition is that the gradient value should be zero or approximately zero.

2.2 Optimization

In the optimization problem of calculating the minimum value in a complex function, we usually use gradient descent method. We can get the minimum value along the gradient direction decreases. In a linear logistic problem, we firstly set a function $h_\theta(x^i) = \theta_1 x^i + \theta_0$. The objective function is

$$\begin{aligned}
J(\theta_0, \theta_1) &= \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 \\
&= \frac{1}{2m}\sum_{i=1}^{m}((\theta_1 x^i + \theta_0) - y^i)^2 \\
&= \frac{1}{2m}\sum_{i=1}^{m}[(\theta_1 x^i + \theta_0)^2 - 2(\theta_1 x^i + \theta_0) + (y^i)^2] \\
&= \frac{1}{2m}\sum_{i=1}^{m}[(\theta_1^2(x^i)^2 + 2\theta_0\theta_1 x^i + \theta_0^2) - 2(\theta_1 x^i + \theta_0) + (y^i)^2]
\end{aligned}$$

where $i = 1, 2, ..., m$, and m is the numbers of samples. We can see that the objective function here is the mean value of squares of all the sample error. Then we use gradient descent to do the optimization. We should firstly calculate the derivation of the objective function, that is,

$$\begin{aligned}
\frac{\partial J(\theta_0, \theta_1)}{\partial\theta_0} &= \frac{1}{2m}\sum_{i=1}^{m}(2\theta_1 x^i + 2\theta_0) - 2y^i) \\
&= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)
\end{aligned}$$

$$\begin{aligned}\frac{\partial J(\theta_0,\theta_1)}{\partial \theta_1} &= \frac{1}{2m}\sum_{i=1}^{m}(2\theta_1(x^i)^2 + 2\theta_0 x^i - 2x^i y^i) \\ &= \frac{1}{m}\sum_{i=1}^{m}(\theta_1 x^i + \theta_0 - y^i)x^i \\ &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x^i\end{aligned}$$

Therefore, the derivation can be summarized as

$$\frac{\partial J(\theta_0,\theta_1)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i$$

where $i = 1, 2, ..., m$; $j = 0, 1$ and m is the numbers of samples. Then we use the samples above to update parameters, that is

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

where α is the step size of each descend.

2.3 Example

From Fig.1, we set a univariate function $J(w, b)$. We firstly take a point on the curve and make a tangent at that point. Then we will get its derivation $\frac{dJ(w)}{dw}$. This derivation is gradient. The objective function changes fastest along the direction of the curve derivative. Then we calculate a w value which decrease a certain length along the opposite direction of the derivation, so $w = w - \alpha\frac{dJ(w)}{dw}$. After iterations, we will find an optimal solution on it.

3 Experimental Evaluation

3.1 Algorithm

We firstly initialize coefficients w_1, w_2 and b_2 to zero. The step size α is a small value, so we set it 2. When the learning rate is extremely small, better weight vectors can be trained. However, a smaller learning rate also means longer training time, and if it is a non-convex problem, it may fall into a local solution, so we do not set a very small value here. An epoch is a process when a complete data set passes the neural network once and return once. We are using a limited data set to do the gradient descent process, so it is not enough to update the weights in only a few updates. To get a good fit, we set epochs 100. For the original data, we set two variables x_1 and x_2. Then we use the MATLAB built-in random function to let them be two arrays, and the lengths are both 200. Then z can be the function $f(x_1, x_2)$.

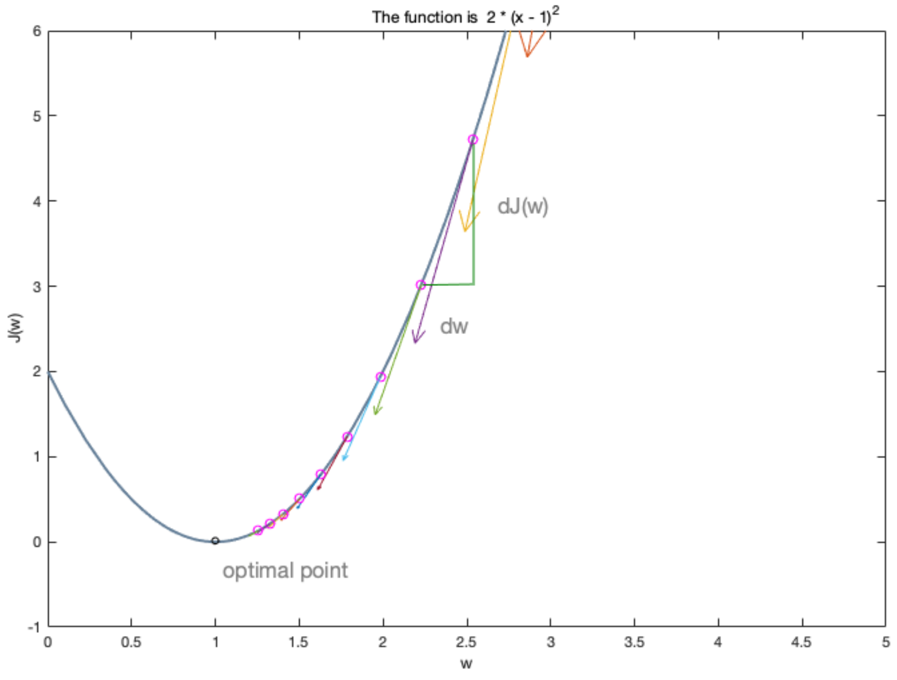

Figure 1: Example of gradient descent process of univariate function in MATLAB

In the algorithm architecture, there are mainly two parts: cost function and gradient descent. We have called these two functions in the main file. We have known that the cost function can be shown as

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

, where $h_\theta(x) = \theta_0 + \theta_1 x^1 + \theta_2 x^2$. The iteration times in the cost function is the length of the original array. For the gradient descent part, there are two circulations in the function. The first level is to calculate the gradients of these three coefficients. Then the second level is to update the coefficient. As we have mentioned before, we should use the gradient descent formula $w = w - \theta \frac{dJ(w)}{dw}$.

3.2 Simulation Result

Initially, we set x and y as two arrays, and their lengths are both 200. The function of z is $z = \frac{1}{2}x + \frac{1}{2}y - 0.5$. After calling the two functions in the executable main file, we plot two figure: one is about the final fitting line, and another is about the cost function.

3.3 Evaluation

From figure.2, we can see that the fitting line is good. Besides, the mean value of the calculated coefficients is approximate to the original ones. From fig.3., we can see that loss values are convergent to zero along with epoch times. Actually, we find that the convergent result can be influenced by learning results and the sample size. For instance, from figure.4, when we set learning rate 0.001, the convergent effect is not as good as the one where the learning rate is 2. We should not initially set learning rate to an extremely value. We can firstly set a relatively small value, when the loss does not decrease, we can use a smaller one.

Besides, excessive sample size will also cause loss to not converge. If the database is too small, it will not cause convergence problems. A small number of samples may only bring about the problem of overfitting. If the training error converges and the verification error does not converge, it is overfitting.

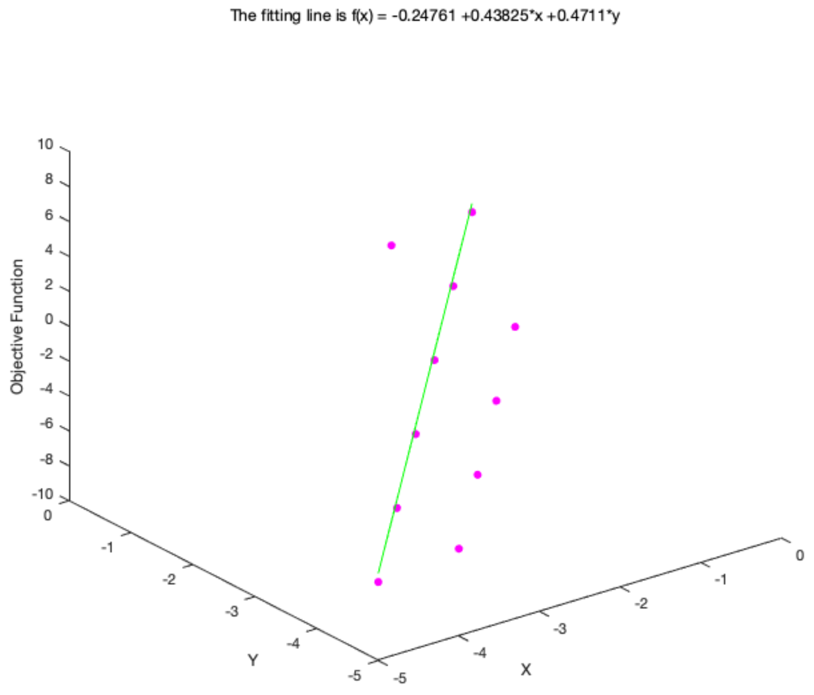

Figure 2: The output fitting line and original data set

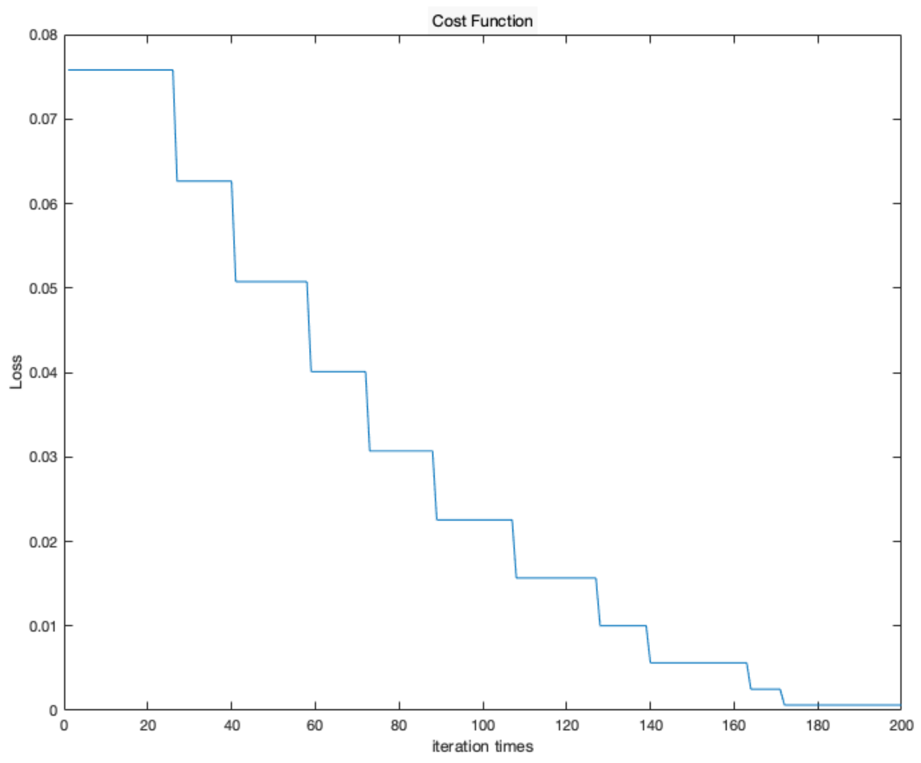

Figure 3: Figure of loss with the number of iterations (α=2, sample numbers=200)

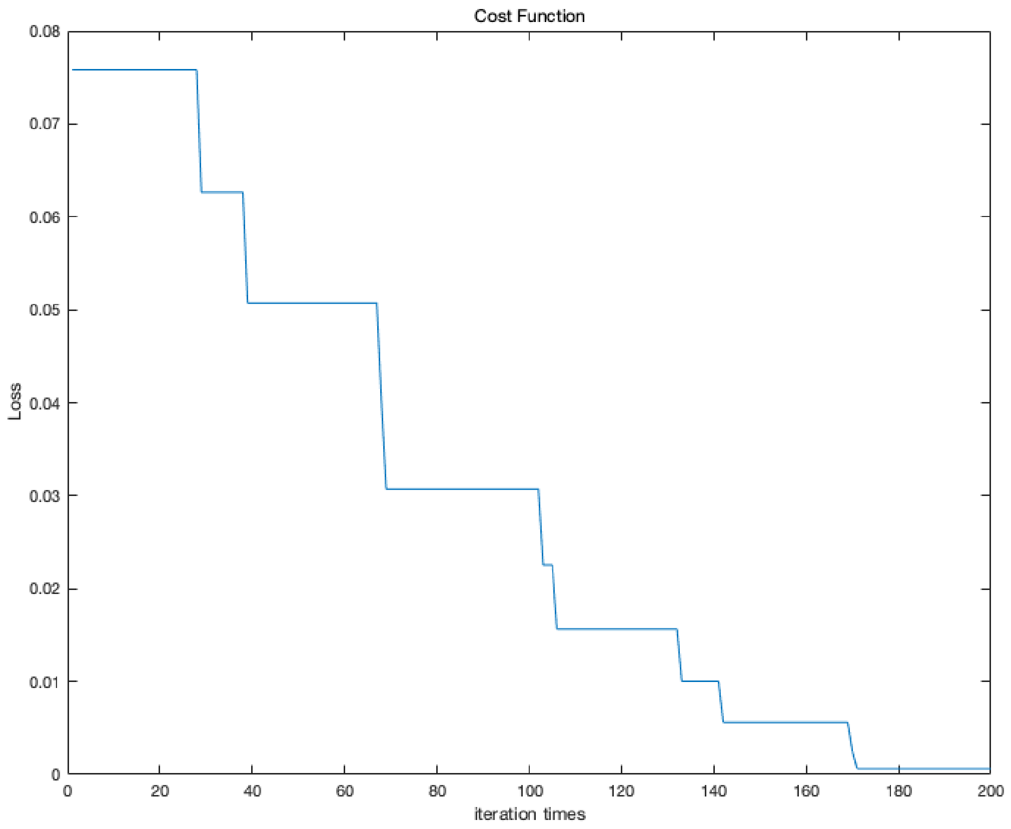

Figure 4: Figure of loss with the number of iterations (α=0.001, sample numbers=200)