Gradient descent

Phuong Luu Vo

December 14, 2022

Most of the optimization problems do not have a closed-form solution. Therefore, **iterative methods** are commonly used to find solutions to convex optimization problems.

In this lecture, we consider *first order algorithms* including gradient descent, stochastic gradient descent, subgradient, and proximal gradient descent to solve an unconstrained convex optimization problem.

1 Introduction

Consider the unconstrained optimization problem

$$\min_x f(x)$$

where f is convex and differentiable for all $x \in dom(f)$.

The necessary and sufficient condition for x^* be the optimal point is

$$\nabla f(x^*) = 0.$$

The iterative method generates a sequence $x^{(0)}, x^{(1)}, \ldots, x^{(k)}, \ldots \in \mathbf{dom} f$ such that $\nabla f(x^*) \to 0$ when $k \to \infty$. Therefore, $f(x^{(k)}) \to p^*$, which is the optimal solution.

1.1 General descent method

The update

$$x^{(k+1)} = x^{(k)} + t_k \Delta x^{(k)}$$

such that $f(x^{(k+1)}) < f(x^{(k)})$ for any $k = 0, 1, ...$ is called *descent method.* $\Delta x^{(k)}$ is *search direction* and t_k is *line search.*

From first-order condition,

$$f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

Replacing $y = x^{(k+1)}$ and $x = x^{(k)}$ yields

$$f(x^{(k+1)}) \geq f(x^{(k)}) + t_k \nabla f(x^{(k)})^T \Delta x^{(k)}.$$

To guarantee $f(x^{(k+1)}) < f(x^{(k)})$, it must to be

$$\nabla f(x^{(k)})^T \Delta x^{(k)} < 0. \tag{1}$$

Vector $\nabla f(x^k)$ and search direction Δx^k form an obtuse angle. Or search direction and $\nabla f(x)$ are in different halfspaces created by the hyperplane $\nabla f(x^k)^T(x - x^k) = 0$ (see Fig. 3).

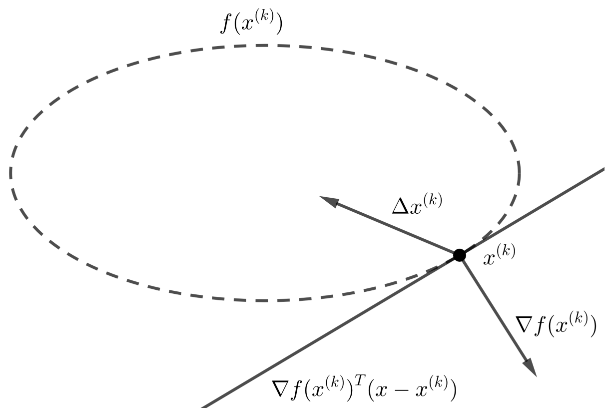

Figure 1: *Search direction* in descent method. Δx and $\nabla f(x)$ are in different halfspace.

General descent method is as follows:

Algorithm 1 General descent method.

Initialize $x^{(0)} \in \mathbf{dom} f$;
repeat

1. Determine *search direction* $\Delta x^{(k)}$;
2. *Line search*: choose step size $t_k > 0$.
3. Update: $x^{(k+1)} = x^{(k)} + t_k \Delta x^{(k)}$, $k = 1, 2, 3...$

until stopping criterion satisfies.

Theoretically, stopping condition is $\|\nabla f(x)\|_2 \leq \epsilon$, in which ϵ is a very small number.

Choosing step size:

- *fix step size*: the simplest strategy
- *exact line search*: in every iteration, choose t such that

$$t = \text{argmin}_{t>0} f(x + t\Delta x).$$

This is not practical, and almost not used. However, in some special instances that the solution to the problem $\text{argmin}_{t>0} f(x + t\Delta x)$ is easily calculated, we can apply exact line search.

- *backtracking line search*: fix parameters $\alpha \in (0, 1/2)$, $\beta \in (0, 1)$ and $t = 1$, repeat $t := \beta t$ until $f(x + t\Delta x) \leq f(x) + \alpha t \nabla f(x)^T \Delta x$.

2 Gradient descent method (GD)

In *gradient descent* method (GD), search direction $\Delta x = -\nabla f(x)$. Therefore, gradient descent algorithm is as follows:

Algorithm 2 Gradient descent algorithm.

---

initialize $x^{(0)} \in \textbf{dom} f$;
repeat

1. *Line search*: choose step size t_k.
2. Update: $x^{(k+1)} = x^{(k)} - t_k \nabla f(x^{(k)})$, $k = 1, 2, ...$

until stopping criterion satisfies.

Theoretically, the stopping criterion is when $\|\nabla f(x)\|_2 \leq \epsilon$. The practical other criterion is $\|x^{(k)} - x^{(k-1)}\|_2 \leq \epsilon$.

2.1 Convergence analysis of GD

A function f is *Lipschitz continuous* if there is L such that

$$\|f(x) - f(y)\|_2 \leq L\|x - y\|_2. \tag{2}$$

If f is convex and differentiable with $dom(f) = \mathbb{R}^n$, and additionally ∇f is L-Lipschitz continuous, GD converges with fixed step size $t \leq 1/L$,

$$f(x^{(k)}) - p^* \leq \frac{\|x^{(0)} - x^*\|_2^2}{2tk} = \frac{C}{k}.$$

The same result holds for backtracking line search with $t = \beta/L$.

If f is strongly convex, *i.e.*, $f(x) - \frac{m}{2}\|x\|_2^2$ i convex for some $m > 0$, GD with fix step size $t \leq 2/(m + L)$ or with backtracking line search satisfies

$$f(x^{(k)}) - p^* \leq c^k \frac{L}{2} \|x^{(0)}) - p^*\|_2^2,$$

in which $0 < c < 1$ depending on m, $x^{(0)}$.

3 Gradient descent applied in some problems

3.1 Linear regression

The least-square problem in linear regression is given by

$$\text{min. } L(w) = \sum_{i=1}^{N} (y_i - x_i^T w)^2 = ||y - Xw||_2^2. \tag{3}$$

in which $X = \begin{bmatrix} x_1^T \\ ... \\ x_N^T \end{bmatrix}$ and $y = \begin{bmatrix} y_1 \\ ... \\ y_N \end{bmatrix}$ (N data points).

Gradient of $L(w)$ is

$$\nabla L(w) = 2 \sum_{i=1}^{N} x_i (x_i^T w - y_i) = 2(X^T X w - X^T y)$$

Gradient descent algorithm is as follows:

Algorithm 3 Gradient descent applied in least square.

---

initialize w;
repeat

1. choose step size t.
2. update:

$$w := w - 2t \sum_{i=1}^{N} x_i (x_i^T w - y_i)$$

   in matrix form

$$w := w - 2t(X^T X w - X^T y)$$

until stopping condition satisfies.

3.2 Logistic regression

Binary classification using logistic regression is as follows:

$$\text{max.} J(w) = \sum_{i=1}^{N} J_i(w) = \sum_{i=1}^{N} y_i x_i^T w - \log(1 + e^{x_i^T w}),$$

We have gradient

$$\begin{aligned} J_i(w) &= y_i x_i - \frac{e^{x_i^T w}}{1 + e^{x_i^T w}} * x_i \\ &= (y_i - \sigma_i) x_i, \end{aligned}$$

in which $\sigma_i = \frac{e^{x_i^T w}}{1+e^{x_i^T w}}$.

Gradient descent applied in logistic regression is as follows:

Algorithm 4 Gradient descent applied in logistic regression.

---

initialize w;

repeat

1. choose step size t.
2. update:

$$w := w + t \sum_{i=1}^{N} (y_i - \sigma_i) x_i$$

until converge.

Some notes:

- In above example, the optimization is maximizing the log likelihood, hence, gradient update must be

$$w := w + t \nabla J(w). \tag{4}$$

- In many applications, especially in deep learning, gradient is calculated numerically. E.g., backward function in PyTorch can calculate the gradient if x is a tensor with $requires_grad = True$.

The following code implements GD in logistic regression from scratch with artificial dataset.

Import library:

```
import numpy as np
import matplotlib.pyplot as plt
```

Define sigmoid function:

```
def sigmoid(s):
  return 1/(1+ np.exp(-s))
```

Generate dataset with N data points, D parameters. X with size $N \times D$ follows normal distribution. Y is generated using logistic model with *w_true* plus some noise.

```
np.random.seed(1)
D = 10      # number of parameters
N = 100   # number of samples
w_true = np.random.randn(D,1)
X = np.random.normal(0, 5, size=(N,D))
Y = np.round(sigmoid(X.dot(w_true) \
                + np.random.normal(0,1,size=(N,1))))
```

Verify sizes of X, Y

```python
print(X.shape)
print(Y.shape)
```

```
(100, 10)
(100, 1)
```

Gradient of J_i

```python
def gradient_i(w, xi, yi):
  sigma_i = sigmoid(np.dot(xi,w))
  return xi*(yi - sigma_i)
```

Step size, number of iterations and initialize w

```python
step_size = 0.005
iters = 100

w_init = np.random.rand(D, 1)
w = w_init.T
obj=np.array([])
w_last = w_init
count = 0
```

obj, w store the values of objective and w for all iterations to visualize the convergence of GD.

Gradient descent update:

```python
while count < iters:
  # calculate gradient
  grad = np.zeros((D,1))
  for i in range(N):
    xi = X[i]
    yi = Y[i]
    grad += gradient_i(w_last, xi, yi).reshape(D,1)

  # perform gradient update and append to w
  w_cur = w_last + step_size * grad
  w = np.vstack((w, w_cur.T))

  # calculate objective value and append to obj
  obj_cur = np.sum(Y*np.dot(X,w_cur) - \
                   np.log(1+np.exp(np.dot(X,w_cur))), axis=0)
  obj = np.append(obj,obj_cur)

  w_last = w_cur
  count += 1
```

We can also calculate the objective by the sum of J_i:

```
obj_cur = 0
for i in range(N):
  xi = X[i]
  yi = Y[i]
  obj_cur += yi*np.dot(xi.T,w_cur) - \
               np.log(1+np.exp(np.dot(xi.T,w_cur)))
```

Gradient J is calculated in matrix form:

```
sigma = sigmoid(np.dot(X,w))
grad = np.sum(X*(Y-sigma), axis=0)}
```

Print the last value print("w*",w_last).

Visualize:

```
plt.plot(obj, label="step_size: {}".format(step_size))
plt.xlabel("iterations")
plt.ylabel("log likelihood")
plt.legend()
plt.savefig('./GD_obj_s{}.png'.format(step_size))
```

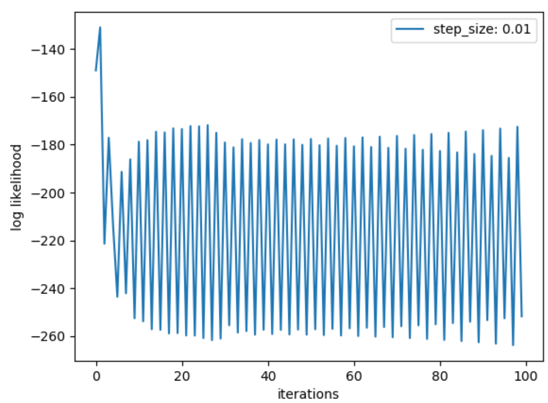

Figure 2: GD with step_size = 0.01.

References

[1] Stephen P. Boyd and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2004.

[2] Convex optimization lectures. Available at www.stat.cmu.edu/∼ryantibs/convexopt/.

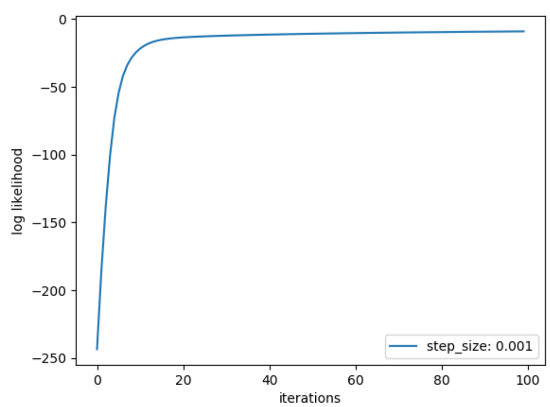

Figure 3: GD with step_size $= 0.001$.