MATH 728D: Machine Learning Lab #5: Optimization

John Burkardt

December 5, 2018

When does $f(x) = 4x^3 - 16x$ attain its minimum value?

This question is an example of the *optimization problem*, which seeks input values x for which a function attains a minimum or maximum value. We ask this question, because often, the function $f(x)$ is measuring a cost, or an error, and it is valuable to know how to minimize such things.

This lab looks at the gradient descent method for minimization.

1 Gradient Descent in 1D

Consider the function $f(x) = 2x^4 - 4x^2 + x + 20$ and suppose that we seek a value x for which this function is minimized. Moreover, we suspect that this value lies in the interval $-2 \leq x \leq 2$.

Our approach will require us to have available the derivative function $f'(x)$, which we will use in order to gradually improve our initial guess for the minimizer.

Exercise 1::

1. write `function fx = f(x)` to evaluate $f(x)$;
2. write `function fpx = fp(x)`to evaluate $f'(x)$;
3. choose a random initial guess `x` in $[-2, +2]$;
4. choose a small `r`, a "learning rate", maybe 0.05;
5. choose a small derivative tolerance `tol`, maybe 0.001;
6. while $tol < |fp(x)|$:
   (a) update `x = x - r * fp(x)`;
   (b) print `x, f(x), fp(x)`;
7. report final x, fx, fp
8. plot $f(x)$ for $-2 \leq x \leq +2$:

If you run the program several times, do you get roughly the same result? Does the plot suggest that you have found the minimizer of $f(x)$ in [-2,+2]?

2 Gradient Descent in Higher Dimensions

Now suppose $f(x)$ is a function of a n-dimensional argument x, which we will think of as a row vector. Then the derivative information we seek is known as the *gradient*, symbolized by ∇f, and also regarded as a row vector:

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots \frac{\partial f}{\partial x_n} \right]$$

It is simple to adjust our gradient descent algorithm for the n-dimensional case.

Consider the function:

$$f(x) = 2x_1^2 - 1.05x_1^4 + x_1^6/6 + x_1 x_2 + x_2^2$$

Exercise 2::

1. write `function fx = f(x)` to evaluate $f(x)$;
2. write `function fpx = fp(x)`to evaluate $\nabla f(x)$;
3. Use `-3+6*rand(1,2)` to choose an initial guess `x` in $[-3,+3] \times [-3,+3]$;
4. choose a small `r`, a "learning rate", maybe 0.05;
5. choose a small derivative tolerance `tol`, maybe 0.001;
6. while $tol < ||fp(x)||$:
   (a) update `x = x - r * fp(x)`;
   (b) print `x, f(x), fp(x)`;
7. report final x, fx, fp
8. surface plot $f(x)$ for $[-2,+2] \times [-2,+2]$:
9. contour plot $f(x)$ for $[-2,+2] \times [-2,+2]$:

Do your plots suggest that you have found the minimizer?

3 Gradient Descent for Model Parameters

Instead of a formula $f(x)$, many optimization problems start with a mass of data to be approximated by a simple model. For instance, we may have n pairs of values (x_i, y_i). We are looking for parameters b and m in a linear model of the form $y = b + m * x$ that approximates our data well.

Obviously, b and m are our unknowns. For any particular i, we will measure the error E_i as the square of the difference between the model's prediction based on x_i, and the actual data y_i:

$$E_i(b, m; x_i, y_i) = (y_i - b - mx_i)^2$$

Then our total error is simply:

$$E(b, m; x, y) = \sum_{i=1}^{n} E_i(b, m; x_i, y_i)$$

To use the gradient method, we will need to compute the vector

$$\nabla E(b, m; x, y) = \left[\frac{\partial E}{\partial b}; \frac{\partial E}{\partial m}\right]$$

For this exercise, use the data file *ford_data.csv*, which contains the model year $year_i$, mileage x_i, and selling price y_i for 23 Ford Escorts. We want to approximate the information in this data set by a formula that predicts the price based on mileage.

The x and y variables have quite different scales. If we do not account for this, the gradient descent method will have trouble converging. The simplest thing to do is to normalize both sets of data , so that

$$x \leftarrow \frac{x - \min(x)}{\max(x) - \min(x)} \tag{1}$$

$$y \leftarrow \frac{y - \min(y)}{\max(y) - \min(y)} \tag{2}$$

so that both normalized variables range from 0 to 1.

Exercise 3::

1. Use `csvread()` to read data;
2. Copy column 2 and 3 of data into x and y arrays;
3. Normalize x and y;
4. write `function value = e(b,m,x,y)`;
5. write `function grad = ep(b,m,x,y)` for $\nabla E(b,m,x,y)$;
6. Initialize $b = 0.5$ and $m = 0$;
7. choose a small `r`, a "learning rate", maybe 0.01;
8. choose a small derivative tolerance `tol`, maybe 0.001;
9. while $tol < ||ep(b,m,x,y)||$:
    (a) update `[b;m] = [b;m] - r * ep(b,m,x,y`;
10. report final `b, m, e, ep`.
11. plot the normalized (x, y) data with the line $y = b+mx$ using the parameter values you have computed.

As you debug your code, you may find it helpful to:

- check your $ep(b,m,x,y)$ function against a finite difference approximation;
- limit the iterations to no more than 500;
- print each iteration index, values of b, m and $e(bm,mx,y)$

4 Stochastic Gradient Descent

We repeat the previous problem, but now we use stochastic gradient descent as our optimization method. This time, we still seek to minimize the error function $E(m,b;x,y)$, but each step involves the gradient of a single error function $E_i(m,b;x,y,i)$.

Since we have

$$E_i(b,m;x_i,y_i) = (y_i - b - mx_i)^2$$

then the gradient ∇E_i is simply:

$$\frac{\partial E_i}{\partial b} = -\,2(y_i - b - mx_i) \tag{3}$$

$$\frac{\partial E_i}{\partial m} = -\,2x_i(y_i - b - mx_i) \tag{4}$$

and in the program we will call this `epi`.

Exercise 4::

1. Copy column 2 and 3 of data into x and y arrays;
2. Set `n=length(x`, to count the number of variables;
3. Normalize x and y;
4. write `function value = e(b,m,x,y)` (same as before);
5. write `function gradi = epi(b,m,x,y)` for $\nabla E(b,m,x,y,i)$;
6. Initialize $b = 0.5$ and $m = 0$;
7. choose a small `r`, a "learning rate", maybe 0.01;
8. choose a small derivative tolerance `tol`, maybe 0.001;
9. repeat the following up to n*500 times:

    (a) use `randi([1,n])` to pick a random index `i`;
    (b) compute gradient vector `gradi`;
    (c) update `b = b - r * gradi(1)`, `m = m - r * gradi(2)`;

10. report final `b, m, e, ep`.