8 Gradient based optimization

8.1 Gradient descent

In calculus, the derivative of a function $y = f(x)$ is defined as a the limit,

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

For a differentiable function, the derivative tells us how much the function $y = f(x)$ will change in proportion to an infinitesimal change in x. Geometrically, the derivative is the slope of the tangent line of the function at the point x. The derivative of a simple function at different locations is visualized in figure 8.1.

In optimization problems, we are interested in finding an optimal value x for a given function $f(x)$. Depending on the problem formulation, the optimum can either be defined as a maximum or a minimum value. In the following, we will focus on the minimization problem, but we note that a maximization problem can easily be converted to a minimization problem, for example by taking the negative or inverse of the objective.

In some cases we can compute the derivative of our objective function and set it equal to zero to solve the minimization problem. However, in many practical cases the objective function is too complicated to allow us to easily solve the minimization problem analytically. In those cases, we can instead use a simple *gradient descent* method. Gradient descent is an interative procedure that finds a minimum of a function by taking steps in the x variable proportional to the negative of the derivative of the objective function.

Intuitively, if the gradient of a function $f(x)$ is positive at a

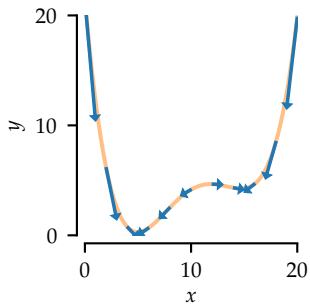

Figure 8.1: Visualization of the derivatives of a fourth order polynomial at different points. Here the derivative of y with respect to x is a scalar that tells us how much y will change for a given change in x. The derivative is shown for different values of x as an arrow from a point (x, y) to a point $(x + \Delta x, y + \Delta y)$ where $\frac{\Delta y}{\Delta x} \propto \frac{dy}{dx}$.

point x, then the function will have a positive slope. Thus, a minimum of the function must be located somewhere in the negative x-direction, and we subtract from x a value proportional to the derivative.

8.1.1 *Gradients of multivariable functions*

If the function we wish to minimize depends on several variables $f(x_1, x_2, \ldots, x_M)$ we need to compute the derivative with respect to each of the input variables. To simplify the notation, we can collect the input variables in a vector (of length M), which we will denote by $\mathbf{x} = [x_1, x_2, \ldots, x_M]^\top$.

We can compute the derivative of the multivariable function $f(\mathbf{x})$ with respect to any single one of its inputs (say x_m) and keep all other inputs fixed. This is known as the *partial derivative* and is most often denoted by $\frac{\partial f}{\partial x_m}$. Based on this, we define the *gradient* of the function as the vector containing all the partial derivatives with respect to each of the inputs,

$$\nabla f(\mathbf{x}) = \frac{df}{d\mathbf{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_M}\right].$$

Example 8.1 Gradient of a simple neural network

Consider a non-linear regression problem where we want to minimize the squared error between a chosen function $f(x)$ and our observations y. Say our function is given by

$$f(x) = \tanh(\theta_0 + \theta_1 x)$$

and the cost function, which is our objective for minimization, is given by

$$C(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\big(y_i - f(x_i)\big)^2.$$

Note that now x are our (fixed) inputs and $\boldsymbol{\theta}$ are the parameters of the function we wish to optimize. The partial derivatives with respect to the parameters θ_0 and θ_1 are

given by

$$\frac{\partial C}{\partial \theta_0} = -\frac{2}{N}\sum_{i=1}^{N}\big(y_i - f(x_i)\big)\big(1 - f^2(x_i)\big),$$
$$\frac{\partial C}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N}\big(y_i - f(x_i)\big)\big(1 - f^2(x_i)\big)x_i,$$

where we have used the fact that $\frac{\mathrm{d}}{\mathrm{d}x}\tanh(x) = 1 - \tanh^2(x)$. Finally, the gradient is given as the vector containing these two partial derivatives,

$$\nabla C(\boldsymbol{\theta}) = \left[\frac{\partial C}{\partial \theta_0}, \frac{\partial C}{\partial \theta_1}\right].$$

8.1.2 *The gradient descent algorithm*

When applied to the minimization of a multivariable function, the gradient descent algorithms works by taking steps in the input argument $\mathbf{x}$ proportional to the negative of the gradient. The size of the steps is controlled by a parameter α.

Definition 8.1 Gradient descent

The method of gradient descent is used to find a minimum of a function $f(\mathbf{x})$ by iterating the following update until convergence.

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \cdot \nabla f(\mathbf{x}^{(t)})$$

Here, $\mathbf{x}^{(t)}$ denotes the value of $\mathbf{x}$ at iteration t, $\mathbf{x}^{(0)}$ is a suitably chosen initial value, and α is a step size parameter.

It is very important to choose the step-size parameter carefully. If the step-size is too large, the updates will overshoot and diverge. If, on the other hand, the step-size is too small, the updates will only make very small changes to $\mathbf{x}$ and the algorithm will take a very long time to converge. An example of the gradient descent algorithm is illustrated in Figure 8.2.

If the objective function has more than one minimum, the gradient descent algorithm not not necessarily converge to the *global* minimum (the value for which the function takes is minimum across its entire domain). Rather, we say that the algorithm finds a *local minimum*, and which one it finds depends on the initial conditions and the step size. Furthermore, if the

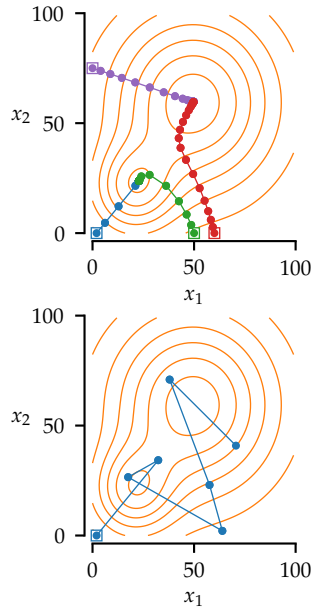

Figure 8.2: Gradient descent on a funtion in two variables. The objective function is illustrated with contour lines (levels sets where the function value is constant). The top plot shows the convergence of gradient descent with a suitably small step size. The algorithm is run from four different initial points marked with squares, and different local minima are found depending on initial conditions. The bottom plot shows six iterations of gradient descent with a fairly large step size.

objective function contains a *saddle-point* (a point where the gradient is zero which is not a local minimum) the gradient descent algorithm might also converge to this.

Under certain mild assumptions about the objective function, the gradient descent algorithm is guaranteed to converge if the step-size is small enough. Since the objective function will decrease if we take a sufficiently small step in the direction of the negative gradient, the sequence of function values we visit will be guaranteed to decrease. As we approach a minimum, the function will be more and more flat, and thus the magnitude of the gradient will decrease. This means that as we approach a minimum the algorithm will take smaller and smaller steps, eventually converging at the minimum.

8.2 Stochastic gradient descent

A disadvantage of the gradient descent algorithm is that it requires us to evaluate the gradient of the objective in each iteration. If we want to estimate parameters in a model based on a large dataset, we need to do computations on the whole data in each iteration, which might not be feasible.

Stochastic gradient descent (SGD) is a simple method that avoids this problem. Instead of computing the exact gradient, in SGD the gradient is estimated based on a small subset of the data. By using a different subset of the data in each iteration, we obtain a "noisy" estimate of the gradient, and while each update of the algorithm does not necessarily reduce the objective function, on average the updates will take us in the direction of a local minimum. To ensure that the SGD algorithm converges, it is common practice to start with a relatively large step-size, and reduce it gradually as the algorithm runs. If the step-size is not reduced, the algorithm will never converge, and will continue to jump around following the noisy gradient estimates.

In addition to reducing the computational burden, SGD also has another advantage. If the objective function has multiple local minima, SGD is not as likely to be trapped in a sub-optimal local minimum as a standard gradient descent algorithm. Intuitively, SGD with its noisy gradient estimates have a chance of escaping a bad local optimum.

Problems

1. What is the gradient of the following function

$$f(x_1, x_2) = x_1^2 \cdot \log(x_1 \cdot x_2)$$

2. If we start at $\mathbf{x}^{(0)} = [2, 2]^\top$ and run the gradient descent algorithm with $\alpha = 0.5$, to minimize the function

$$f(\mathbf{x}) = x_1^2 + 5 \cdot x_2^2$$

   what will the value $\mathbf{x}^{(1)}$ be (after the first iteration)?

3. Is the gradient descent algorithm guaranteed to converge?

4. If you were to run the following gradient descent algorithm

$$x^{(t+1)} = x^{(t)} - \alpha \cdot (2x - 6)$$

   with a suitably small value of α, to which value of x would it converge?

Solutions

1. $\nabla f = \left[2x_1 \log(x_1 \cdot x_2) + x_1, \frac{x_1^2}{x_2}\right]$
2. $\mathbf{x}^{(1)} = [0, -8]$
3. No, if the step-size is too high, the algorithm might diverge.
4. x=3