Gradient

The **gradient** (or gradient vector field) of a scalar function $f(x_1, x_2, x_3, \ldots, x_n)$is denoted ∇f or

$\vec{\nabla} f$ where ∇ (the nabla symbol) denotes the vector differential operator, del. The notation

"grad(f)" is also commonly used for the gradient. The gradient of f is defined as the unique vector field whose dot product with any vector v at each point x is the directional derivative of f along v. That is,

$$(\nabla f(x)) \cdot \mathbf{v} = D_{\mathbf{v}} f(x).$$

in a rectangular coordinate system, the gradient is the vector field whose components are the partial derivatives of f:

$$\nabla f = \frac{\partial f}{\partial x1}\mathbf{e}_1 + \cdots + \frac{\partial f}{\partial xn}\mathbf{e}_n$$

where the e_i are the orthogonal unit vectors pointing in the coordinate directions. When a function also depends on a parameter such as time, the gradient often refers simply to the vector of its spatial derivatives only.

Gradient Descent

> Gradient descent is a first-order iterative optimization algorithm. To find a **local minimum** of a function using gradient descent, one takes steps proportional to the **negative of the gradient** (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

Gradient descent is based on the observation that if the multi-variable function $F(\mathbf{x})$ is defined

and differentiable in a neighborhood of a point $\mathbf{a}$, then $F(\mathbf{x})$decreases fastest if one goes from

$\mathbf{a}$ in the direction of the negative gradient of F at $\mathbf{a}$, $-\nabla F(\mathbf{a})$. It follows that, if

$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$$

forγ small enough, then

$$F(\mathbf{a}) \geq F(\mathbf{b}).$$

In other words, the term $\gamma\nabla F(\mathbf{a})$ is subtracted from $\mathbf{a}$ because we want to move against the gradient, namely down toward the minimum. With this observation in mind, one starts with a guess $\mathbf{x}_0$ for a local minimum of F, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \geq 0.$$

We have

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots,$$

so hopefully the sequence $(\mathbf{x}_n)$ **converges** to the desired local minimum. Note that the value of the step size γ is allowed to change at every iteration. With certain assumptions on the function F (for example, F convex and ∇F Lipschitz) and particular choices of γ (e.g., chosen via a line search that satisfies the Wolfe conditions), convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

Stochastic Gradient Descent

> Stochastic gradient descent (often shortened in SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minimums or maximums by iteration.

The standard gradient descent algorithm updates the parameters θ of the objective $J(\theta)$ as,

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)]$$

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set. Stochastic Gradient Descent (SGD) simply does away with the expectation in the update and computes the gradient of the parameters using only a single or a few training examples. The new update is given by

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x(i), y(i))$$

with a pair $(x(i), y(i))$ from the training set.

Example

Use Linear Regression as Our Example

$$f = h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i$$

Gredient Descent （Batch gradient descent）

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

> Repeat{
>
> $$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y)x_j^{(i)}$$
>
> for every $j = 0, \ldots, n$
>
> }

Stochastic Gradient Descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)} - y^{(i)}))^2$$
$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

```
1. Randomly shuffle training examples
2. Repeat{
       for i := 1,...,m{
           $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)} - y^{(i)})x_j^{(i)}$
           for every $j = 0, \ldots, n$
       }
   }
```

Compare

Previously, we saw that when we are using Batch gradient descent, that is the algorithm that **looks at all the training examples** in time, Batch gradient descent will tend to, you know, take a reasonably straight line trajectory to get to the global minimum like that. In contrast with Stochastic gradient descent every iteration is going to be much faster because we **don't need to sum up over all the training examples**. But every iteration is just trying to fit single training example better.

See More

Optimization: Stochastic Gradient Descent

An overview of gradient descent optimization algorithms

Gredient Descent With Large Datasets