Optimization in Machine Learning

Tyler Chang
April 17, 2019

Convexity

- A function f is *convex* on a region R if its tangent at x, $T_x(y) = f(x) + \nabla f(x)^T(y-x)$, stays "under" f for all $x, y \in R$. I.e., $f(y) \geq T_x(y)$ for all $x, y \in R$.
- A function f is *strongly convex* if any amount of quadratic curvature can be "removed" from f and it remains convex. I.e., $f(y) \geq T_x(y) + \frac{m}{2}\|y-x\|_2^2$ for all $x, y \in R$ for some m.
  - The above can be interpreted as "f can be lower bounded by a quadratic function."
  - Trivially, f can also be bounded above by a quadratic on any closed sublevel set, so $f(y) \leq T_x(y) + \frac{M}{2}\|y-x\|_2^2$ for some $M > m$.
- A function f is convex if its gradient is monotone nondecreasing, and strongly convex if its gradient is monotone increasing in each dimension.
- A function f is convex if its Hessian $\nabla^2 f$ is everywhere *symmetric semi-positive definite* (SPD) denoted: $\nabla^2 f \succeq 0$. This implies that when applied to a vector x, $\nabla^2 f$ cannot "turn x around," i.e., $\langle (\nabla^2 f)x, x\rangle \geq 0$.
- A function f is strongly convex if its Hessian $\nabla^2 f$ is "more SPD" than some pure quadratic: $\nabla^2 f \succeq mI$, for some m.
- A function f is convex if for all $\lambda \in R$, the sublevel set $\{x \in R : f(x) \leq f(\lambda)\}$ is convex as a set.

Conditioning

The rate at which an algorithm can converge will be faster when the sublevel sets of the objective function f are somewhat "square." The *condition number* of a convex set C is defined by $\kappa(C) = \frac{W_{max}^2}{W_{min}^2}$ where W_{max} and W_{min} are the maximum and minimum widths of C respectively. For a function f, its conditioning about a point x is given by the ratio between the smallest and largest eigenvalues (for f symmetric) or between the smallest and largest singular values (for f general) of its Hessian at x:

$$\kappa(f)(x) = \|\nabla^2 f(x)^{-1}\|\|\nabla^2 f(x)\|.$$

Note that for non-quadratic f, the conditioning of f can change at different locations in space.

Rates of Convergence

- A function f converges to a limit L at a *linear rate* if for some $\mu \in (0, 1)$:

$$\lim_{k \to \infty} \frac{|f(x^{(k+1)}) - L|}{|f(x^{(k)}) - L|} \to \mu.$$

  If $\mu = 1$, then the convergence is *super-linear*, and if $\mu = 0$, then it is *sub-linear*.

- By the Big-O notation, $f \to L$ at a rate of $\mathcal{O}(c^k)$ if $|f(x^{(k)}) - L| \leq \mathcal{O}(c^k)$ for $0 < c < 1$.

- Conversely, $f \to L$ at a rate of $\mathcal{O}(\frac{1}{\log \varepsilon})$ if that many iterations are needed to achieve $\mathcal{O}(\varepsilon)$ accuracy.

First Order (Gradient) Methods

For a convex *objective* or *cost* function f, the necessarry and sufficient conditions for optimality are zero gradients. The *gradient descent* update defines a contraction over the gradient ∇f in search of its fixed point $\nabla f(x^\star) = 0$:

$$x^{(k+1)} = x^{(k)} - \alpha \nabla f(x^{(k)}). \tag{1}$$

The constant α is called the *step size* or *learning rate*. For a convex function, the optimal value for α could be determined at each iteration k by performing an *exact line search* over the direction $-\nabla f(x^{(k)})$. Given an exact line search, gradient descent will converge at a linear rate for strongly convex differentiable functions. One could also start with a large step size then perform a *backtracking line search* to avoid overstepping.

If the function f is convex but not differentiable, the same strategy can be used, but instead of a true gradient $\nabla f(x)$, we must settle for a *sub-gradient* $g(x)$, reducing the convergence rate to $\mathcal{O}\left(\frac{1}{\sqrt{t}}\right)$. Similarly, when a *stochastic estimate* g is given that satisfies $\mathbb{E}\left[g(x)\right] = f(x)$ (where $\mathbb{E}$ denotes the expected value of a random variable), the *stochastic gradient descent* (SGD) algorithm is obtained by plugging in g for ∇f in (1).

Due to its non-monotone convergence, SGD will not converge all the way to the true optimum $x^\star$ with a fixed step size. Instead, SGD converges to an elliptical region surrounding $x^\star$ whose *radius of convergence* is dependent on α. So, to converge all the way to $x^\star$, α must be decayed. One common method for doing so is the square summable but not summable rule $\alpha_k = \frac{C_0}{k}$ for some constant C_0. However, for faster convergence, it is best to keep α_k constant until convergence stalls, then decrease α_k only if greater accuracy is desired.

SGD and gradient descent are only guaranteed convergence for convex functions. When applied to nonconvex functions, both get stuck at *local minima*, which may not correspond to the *global minimum*. One common modification to the standard SGD recipe is the addition of *momentum*, which allows the gradient to accelerate toward global minima, potentially plowing through local minima. For a stochastic or ill-conditioned function, momentum

smooths over oscillating gradients, making the path to the minimum much more direct. For a momentum coefficient $\beta \in (0, 1)$, we replace the gradient $\nabla f(x^{(k)})$ in (1) with the modified:

$$g^{(k)} = \beta g^{(k-1)} + (1 - \beta)\nabla f(x^{(k)})$$

where in an abuse of notation, $\nabla f(x^{(k)})$ could denote either the true gradient or a stochastic estimate (for SGD). Large values of β lead to smoother convergence at the risk of causing divergence for large step sizes, so it is typical to set $\beta \approx 1$ and use the largest convergent step size.

An alternative to the above momentum, is *Nesterov's momentum*, which is essentially a vector accelleration technique for *vanishing gradients*. Another smoothing technique that is often used is the *proximal point* algorithm, where a nonconvex objective f is decomposed into $f = g+h$, where g is smooth and convex and h is noisy and nonconvex. Then we can converge much faster by alternating between optimizing an *envelope function* $E(x) = g(x) + R_1(x)$ and a *proximal operator* $P(x) = h(x) + R_2(x)$, where R_1 and R_2 are regularization terms that penalize overstepping.

First order methods are characterized by iteratively minimizing a first order Taylor expansion (i.e., $f(y) \approx f(x) + \nabla f(x)^T y$), subject to a quadratic penalty on overstepping (based on α). On a final note, there exist pathologically hard convex problems, for which no first order method can achieve faster than first order (linear) convergence.

Higher Order Methods

The most classic way to account for curvature and poor conditioning is by *second order* (i.e., Hessian based) methods. Newton's method is based off a second order Taylor expansion, and heuristically defines an elliptical trust region about the current iterate $x^{(k)}$ based on the curvature of f, then "jumps" to the minima in that trust region:

$$x^{(k+1)} = x^{(k)} + \nabla^2 f(x^{(k)})^{-1} \nabla f(x^{(k)}). \tag{2}$$

For strongly convex functions, Newton's method is known to have *second order* (quadratic) convergence.

The computation of and inversion of the Hessian $\nabla^2 f(x^{(k)})^{-1}$ can be prohibitively expensive for high-dimensional problems. Therefore, many *Quasi-Newton* methods use approximations to the Hessian. Most famous among these is BFGS which uses a Rank-1 matrix update based on the secant condition to iterate toward the true Hessian. To avoid recomputing the inverse, the Rank-1 *Sherman-Morrison-Woodbury matrix identity* is leveraged for the inverse update. For strongly convex functions, BFGS has super-linear convergence. Unfortunately, for non-convex functions, Quasi-Newton methods exhibit unwanted behaviors. Specifically, they are attracted to saddle points, which occur frequently in non-convex high-dimensional problems.

More recent approaches include the *adaptive gradient* (Adagrad), which replaces the Hessian $\nabla^2 f(x^{(k)})$ in (2) with an iteratively refined estimate G for the variance of f with respect to each basis dimension. Another popular trust region method uses the *Fischer information matrix* instead of G. *ADAM* takes this a step further, using momentum to smooth both the gradient and the variance estimate G.

Lagrangian Duality

All the techniques discussed so far have been *interior point* methods. To extend these algorithms to the constrained case, we could simply project back into the *feasible region* every time a constraint is violated. However, this is expensive and can degrade convergence rates. Often, particularly for objective functions with analytic solutions, the preferred approach is *Lagrangian optimization*.

Consider the following constrained minimization problem with linear constraints:

$$\min f_0(x) \text{ s.t. } f_i(x) \leq 0 \text{ and } h_j(x) = 0 \text{ for } i = 1, ..., m \text{ and } j = 1, ..., n. \tag{3}$$

The above is referred to as the *primal problem*. Its *Lagrangian* is given by

$$L(x, \lambda, v) = f_0 + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{n} v_j h_j(x)$$

where violating the constraints is "penalized" by the *Lagrangian multipliers* λ_i and v_j. The *dual function* is a function of these weights

$$g(\lambda, v) = \inf_x L(x, \lambda, v).$$

Note that the dual problem is always concave in the weights λ and v, so it can be maximized using gradient ascent. Let $x^\star$ be the solution to the primal problem, then $f_0(x^\star) \geq g(\lambda, v)$ for all λ and v. When maximized, the dual solution gives a tight lower bound for the primal solution:

$$g(\lambda^\star, v^\star) \leq f_0(x^\star)$$

where the difference in the inequality is called the *duality gap*.

The property that the dual solution always lower bounds the primal is called *weak duality*. *Strong duality* is the desirable property that in fact $g(\lambda^\star, v^\star) = f_0(x^\star)$. Strong duality does not hold in general, but if f is convex and *Slater's condition* (that the relative interior of the feasible region is nonempty) holds, then strong duality always holds.

The *Karush-Kuhn-Tucker* (KKT) conditions are necessary conditions for optimality and also sufficient under strong duality. Points x and (λ, v) can be primal/dual optimal only if:

- $f_i(x) \leq 0$ for $i = 1, \ldots, m$ (primal feasibility)
- $h_j(x) = 0$ for $j = 1, \ldots, n$ (primal feasibility)
- $\lambda_i \geq 0$ for $i = 1, \ldots, m$ (dual feasibility)
- $\lambda_i f_i(x) = 0$ for $i = 1, \ldots, m$ (complementary slackness)
- $\nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{j=1}^{n} v_j \nabla h_j(x) = 0$ (stationary point/zero gradient condition).

At a primal optimal solution, the dual weights tell us sensitivity information. Specifically, if we relaxed constraint f_i by some small amount, we could expect to see improvement by a factor of λ_i in the optimal value, and similarly for each h_j and v_j.

Note that the minimization of the Lagrangian with respect to each set of dual weights is implicit in the formulation of the dual gradient. Therefore, the Lagrangian form is most useful when it has an analytic solution for fixed weights λ and v, as is the case for linear and quadratic programming problems.

Automatic Differentiation and Backpropogation

To obtain the loss derivative for a neural network with respect to its weights, a form of automatic differentiation called *backpropogation* is used. For a function $f(x)$, backpropogation can be interpreted as the chain rule:

- For each intermediate variable, compute its derivative as a function of other intermediate variables using the chain rule.
- Finally, compute the first intermediate variable's derivative with respect to each x_i and the derivative of f with respect to all intermediate variables.
- If there are no "loops" in the dependency graph, this defines an upper triangular linear system that can be solved via backsubstitution.

Discrete Optimization

The analogue of convexity in the context of discrete optimization is *submodularity*. A function $F : 2^X \to \mathbb{R}$ is submodular if it has a diminishing returns property: for $A, B \subseteq X$, $F(A) + F(B) \geq F(A \cup B)$ (i.e., F is an outer measure). For submodular functions, greedy algorithms are analagous to gradient descent in the continuous case. Alternatively, discrete optimization can be made continuous by applying the *Lovasz extension*, where each valid combination of discrete variables is placed in a lattice and the objective values are linearly interpolated using triangulation.

Bayesian Optimization

Bayesian optimization is a common *global optimization* algorithm. Given the sampled data points D and the current minima f_{min}, one can use Bayes's theorem

$$P(f(X) < f_{min} | \{D\}) = \frac{P(\{D\} | f(X) < f_{min}) P(f(X) < f_{min})}{P(\{D\})}$$

to make an informed guess about where to sample f next to get the most "potential improvement." To do so, we must have some knowledge of the *prior distribution* $P(f(X) < f_{min})$ which we approximate by a Gaussian prior kernel $K_v(x)$ of fixed variance v.

Since finding the best potential improvement via Baye's theorem is expensive, Bayesian optimization is only useful for nonconvex problems whose dominant cost is function evaluations (such as hyperparameter tuning).

Issues in Machine Learning

Often, networks that obtain low training error have high testing or *generalization error*. This is becaues a sufficiently complex neural network has enough degrees of freedom that it can interpolate noise. Generalization error typically corresponds to sharp deep minima in the optimization landscape, where f is poorly conditioned. With SGD, this generalization error can be mitigated by purposely creating a radius of convergence large enough that the algorithm will "bounce out of" sharp minima.

No Free Lunch Theorem

If we know nothing about a function f (not even if it is measurable), then (provably) no optimizer can outperform random sampling in expectation. Therefore, in order to obtain any meaningful convergence guarantees, we must assume *something* about the class of functions we are minimizing. Generally, stronger assumptions will result in faster converging algorithms.