

Language and Statistics II

Lecture 3: Optimization

Noah Smith

September 1, 2009

It's in our nature as computer scientists to find broad classes of problems.

In applied CS, this is a time-saver: use a general solver or implementation to solve one instance.

This lecture: **numerical optimization**.

Three levels of understanding:

- 1 turning a learning or prediction problem into a well-formed optimization problem, and knowing which of a set of algorithms can be called to solve the problem;
- 2 understanding intuitively how each algorithm works and being able to implement it; or
- 3 when faced with a new optimization problem that does not fit existing algorithms, developing a new optimization algorithm that is suitable, or directly improving on runtime or memory efficiency requirements of existing algorithms.

Background

- Huge topic (10-725)
- Studied in math, applied math, business departments (“operations research”)
- General setting:

$$\min_{\mathbf{w} \in \mathcal{W}} \Phi(\mathbf{w}) \quad (1)$$

- $\mathcal{W} \subseteq \mathbb{R}^d$ is a set of possible solutions (encodes *constraints*, e.g., $c(\mathbf{w}) = 0$ or $c(\mathbf{w}) \geq 0$)
- $\Phi : \mathcal{W} \rightarrow \mathbb{R}$

About \mathcal{W}

It might be:

- continuous (as today) or discrete or a mix
- unconstrained: $\mathcal{W} = \mathbb{R}$
- “bounds” constraints
- linear constraints
- convexity

Convexity (sets)

A set $\mathcal{W} \subset \mathbb{R}^d$ is convex if and only if, for every two points w and w' both $\in \mathcal{W}$, the straight line segment connecting w and w' is *also* entirely inside \mathcal{W} .

*

About Φ

It might be:

- linear
- quadratic
- continuous
- (globally) convex
- (globally) differentiable

Convexity (functions)

A function $\Phi : \mathcal{W} \rightarrow \mathbb{R}$ is convex if and only if, for all \mathbf{w} and \mathbf{w}' both $\in \mathcal{W}$, and for all $\alpha \in [0, 1]$,

$$\Phi(\alpha \mathbf{w} + (1 - \alpha) \mathbf{w}') \leq \alpha \Phi(\mathbf{w}) + (1 - \alpha) \Phi(\mathbf{w}') \quad (2)$$

*

Today

Continuous problems without constraints.

Most of the theory assumes the function is convex.

We often deal with functions that are not convex. Loosely speaking, this means the methods that assume convexity will find *local* optima. (But be careful.)

Min vs. Max

We can easily convert from a “min” problem to a “max” one or vice versa. (How?)

Much of the art of optimization involves converting an optimization problem into one you know how to solve, or breaking it into easier parts. (Level 3 stuff.)

Convex programming

Generally means Φ is convex and constraints are either

- Linear equality constraints or
- Concave inequality constraints, written as $c(\mathbf{w}) \geq 0$

Local optima = global optima.

Machine learning fixates on convex programming.

Special case: linear programming

Φ is linear and all constraints are linear (equalities and/or inequalities).

Special, polynomial-time methods exist!

Don't confuse this with *integer* linear programming, which is NP-hard.

Hill-climbing

Input: objective function $\Phi : \mathcal{W} \rightarrow \mathbb{R}$, initial vector $\mathbf{w}^{(0)}$,
maximum iterations t'

Output: final vector \mathbf{w}

$t \leftarrow 0$

repeat

$t \leftarrow t + 1$

$\mathbf{w}^{(t)} \leftarrow \text{ascend}(\mathbf{w}^{(t-1)}, \Phi)$

until convergence, defined as $\mathbf{w}^{(t)} \approx \mathbf{w}^{(t-1)}$,

$\Phi(\mathbf{w}^{(t)}) \approx \Phi(\mathbf{w}^{(t-1)})$, and/or $t \geq t'$

$\mathbf{w} \leftarrow \mathbf{w}^{(t)}$

return \mathbf{w}

A generic hill-climbing algorithm. $\text{ascend} : \mathcal{W} \times (\mathcal{W} \rightarrow \mathbb{R}) \rightarrow \mathcal{W}$ is a subroutine that, when given $\mathbf{w}^{(t-1)}$ and Φ , is assumed to return \mathbf{w} such that $\Phi(\mathbf{w}) \geq \Phi(\mathbf{w}^{(t-1)})$.

If we assume Φ is continuous and differentiable, then we have

- First derivative w.r.t. w_j : $\nabla_{w_j} \Phi : \mathcal{W} \rightarrow \mathbb{R}$
- Gradient vector: $\nabla_{\mathbf{w}} \Phi : \mathcal{W} \rightarrow \mathbb{R}^d$
- Hessian: $\mathbf{H}_{\mathbf{w}} \Phi : \mathcal{W} \rightarrow \mathbb{R}^{d \times d}$ (symmetric matrices)

Some theorems

- FONC: \mathbf{w}_0 is a local minimizer of $\Phi \Rightarrow \nabla_{\mathbf{w}}\Phi(\mathbf{w}_0) = \mathbf{0}$.
- SONC: ... and $\mathbf{H}_{\mathbf{w}}\Phi(\mathbf{w}_0)$ is psd.
- SOSC: $\mathbf{H}_{\mathbf{w}}\Phi$ is continuous and $\nabla_{\mathbf{w}}\Phi(\mathbf{w}_0) = \mathbf{0}$ and $\mathbf{H}_{\mathbf{w}}\Phi(\mathbf{w}_0)$
 $\Rightarrow \mathbf{w}_0$ is a local minimizer of Φ .

Punchline: it'd be nice to have \mathbf{w}_0 such that $\nabla_{\mathbf{w}}\Phi(\mathbf{w}_0) = \mathbf{0}$.

Ascent direction

Idea:

$$\text{ascend}(\mathbf{w}, \Phi) = \mathbf{w} + \alpha \Delta \quad (3)$$

for direction $\Delta \in \mathbb{R}^d$ and step size $\alpha \in \mathbb{R}_{\geq 0}$.

- **Gradient ascent:** use “steepest” ascent direction at \mathbf{w} , which is $\nabla_{\mathbf{w}} \Phi(\mathbf{w})$
Dummy notation: \mathbf{g}
- **Newton’s method:** use $\mathbf{H}_{\mathbf{w}} \Phi(\mathbf{w})^{-1} \nabla_{\mathbf{w}} \Phi(\mathbf{w})$
Dummy notation: $\mathbf{H}^{-1} \mathbf{g}$

Line search

The direction Δ isn't everything; also need to know how far to go.

$$\max_{\alpha \in \mathbb{R}_{\geq 0}} \Phi(\mathbf{w} + \alpha\Delta) \quad (4)$$

Tradeoffs:

- increase Φ a lot, but don't go too far
- don't spend too much time

In practice: various closed forms, sometimes quite simple.

Good and bad news

Easy, if we can calculate gradient (and maybe Hessian).

Slow.

Newton's method: probably too expensive if $d \gg 0$.

Solution

Conjugate gradient methods calculate an update that is *conjugate* to the most recent direction. (Don't do the same work twice.)

Quasi-Newton methods approximate the Hessian, often with just a few vectors. L-BFGS is one used very often these days.

Both of these involve some “memory” of previous state (so think of ascend as having its own memory). Many implementations exist.

Stochastic gradient ascent

Many Φ break down into a summation:

$$\Phi(\mathbf{w}) = \sum_{i=1}^{\tilde{N}} \Phi_i(\mathbf{w}) = \tilde{N} \mathbb{E}_{I \sim \text{unif}(\tilde{N})} [\Phi_I(\mathbf{w})] \quad (5)$$

Pick i uniformly at random. Let

$$\text{ascend}(\mathbf{w}, \Phi) = \mathbf{w} + \alpha \nabla_{\mathbf{w}} \Phi_i(\mathbf{w}) \quad (6)$$

Coordinate ascent

For any $j \in \{1, \dots, d\}$,

$$\Phi(\mathbf{w}) = \Phi_j(\mathbf{w}) + \underbrace{\bar{\Phi}_j(\langle w_1, \dots, w_{j-1}, w_{j+1}, \dots, w_d \rangle)}_{\text{constant}(w_j)} \quad (7)$$

This suggests iterating among the w_j , improving Φ with respect to each in turn. Sometimes a closed form exists or the subproblem is convex.

A bit about constraints

There isn't time to teach you the full theory and practice of Lagrange multipliers. Instead, we'll go through one example.

Consider a d -sided die that is not fair. We have a sequence of training examples $\langle \tilde{x}_1, \dots, \tilde{x}_{\tilde{N}} \rangle$, with each $\tilde{x}_i \in \{1, 2, \dots, d\}$.

The weights $\mathbf{w} = \langle w_1, \dots, w_d \rangle$ must be nonnegative and sum to one.

Steps

- 1 Write down the constrained problem (\mathcal{W}, Φ) .
- 2 Add a Lagrange multiplier for each constraint, and write Ξ and Ψ .
- 3 Invoke $\max_{\mathbf{w}} \min_{\lambda} \Psi(\mathbf{w}, \lambda) = \min_{\lambda} \max_{\mathbf{w}} \Psi(\mathbf{w}, \lambda)$.
- 4 Solve for $\mathbf{w}^*(\lambda)$.
- 5 Optimize $\min_{\lambda} \Psi(\mathbf{w}^*(\lambda), \lambda)$.
- 6 Return $\mathbf{w}^*(\lambda^*)$.

Chew on this

What if we don't know the number of sides (d) of the die?

Can you derive a maximum likelihood estimate for both d and the vector (of unknown length d) \mathbf{w} ?

Closing remarks

- Many details skipped.
- Many implementations to use in many languages.
- Things to watch out for: starting point $\mathbf{w}^{(0)}$ and convergence criterion
- Use a known algorithm when you can, but make sure it's suitable.
- If you can derive your own algorithm, your life will be easier.