Lec 15 SoS

Let $C(h)$ be the coefficient of h in monomial basis, so

$$h(x) = \sum_{\vec{\alpha}:\text{non-negative vectors in } \mathbb{Z}^d} C(h)_{\vec{\alpha}} \prod_{i\in[d]} x_i^{\vec{\alpha}_i}$$

So we need to add the constraint:

$$\sum_{\vec{\alpha}:\text{non-negative vectors in } \mathbb{Z}^d} C(h^2)_{\vec{\alpha}} X_{\vec{\alpha}} \geq 0$$

Theorem

The new minimization problem is equivalent to the original problem, in the sense that

- *(1).* $\min_{X \text{ satisfies all constraints}} G(X) = \min_x f(x)$.
- *(2). As long as f has a unique minimizer x^*, then G has a unique minimizer X^* and*

$$x^* = (X^*_{(1,0,0,\ldots,0)}, X^*_{(0,1,0,\ldots,0)}, \ldots, X^*_{(0,0,0,\ldots,0,1)})$$

$\tilde{\mathbb{E}}_X g = \sum_{\vec{\alpha}} C(g)_{\vec{\alpha}} X_{\vec{\alpha}}$ $\min \tilde{\mathbb{E}}_X f$, s.t. for all polynomial h:$\tilde{\mathbb{E}}_X h^2 \geq 0$

1. G is always linear (convex) 2. Minimizing G can not recover the minimizer of f , because the variables X_α are constraint and dependent 3. For every polynomial h(x), we must have $h(x)^2 \geqslant 0$

4. This is convex optimization over convex constraint set (polytope)

5.Constant degree SoS can always be solved efficiently. Since constant D can be solved by interior point

6.SoS implies that optimizing over a polynomial over a polynomial constraint set can be solved using cvxopt, i.e. any analytic optimization problem can be turned into a convex optimization problem.

7. SoS implies that optimizing over a constant degree polynomial over a constant degree polynomial constraint set can be solved efficiently.

8. Degree 101 SoS over $x \in \mathbb{R}^d$ implies $\tilde{\mathbb{E}}_X x_1^4 \geq 0$, for $x_1 \in \mathbb{R}$ is TRUE, since degree 101 is much larger than 4

9. Degree 2 SoS over $x \in \mathbb{R}^d$ implies $\tilde{\mathbb{E}}_X x^4 \geq 0$, for $x_1 \in \mathbb{R}$ is FALSE, since for degree 2, $M_D(X)$ contains terms of degree at most 2

Lec 16 Momentum

The Gradient Descent Lemma: for $\eta \leq \frac{1}{L}$, $f(x_{t+1}) \leq f(x_t) - \frac{\eta}{2}\|\nabla f(x_t)\|_2^2$

Key idea: always use large learning rate,use weighted sum of the gradients from the previous iterations.

1. Nesterov's Accelerated Gradient Descent (L-smooth function)

Gradient Descent step: $z_{t+1} = x_t - \eta\nabla f(x_t)$, Momentum step: $x_{t+1} = (1-\gamma_t)z_{t+1} + \gamma_t z_t$

Intuitively, it makes Gradient Descent more stable when using large learning rate. Momentum tunes the learning rate automatically according to local geometry. **Key observation**: Gradient is smaller than usual We can now use a larger learning rate without bumping back and forth.

2. Heavy ball/Polyak Momentum (Used in practice)

$x_{t+1} = x_t - \eta g_t$, $g_t = \gamma \sum_{s\leq t} (1-\gamma)^{t-s}\nabla f(x_s)$

Worst case, provably not better than GD. Approximate the Nesterov Accelerated GD, easy to implement

3. Nestrov's Momentum is provably faster than GD to optimize a L-smooth convex functions

4. Polyak's Momentum is NOT provably faster than GD to optimize L-smooth convex functions

5. Polyak's momentum, you should use larger learning rate compared to that of GD.

6. Polyak's momentum, you should increase the momentum factor $1-\gamma$ when the loss is zig-zaging. Because increasing momentum should decrease effective learning rate.

7. Momentum update is different from the GD update with a per-step learning rate scheduling. Since it will reduce the update size on sharp directions and increase the update size on flat directions

3. Linear Coupling

- At every iteration, for a fixed τ:
- Gradient Descent (proper learning rate): $s_{t+1} = x_t - \frac{1}{L}\nabla f(x_t)$.
- Update (large learning rate $\eta \gg \frac{1}{L}$): $l_{t+1} = l_t - \eta\nabla f(x_t)$.
- Linear coupling: for a $\tau \in [0,1]$: $x_{t+1} = (1-\tau)s_{t+1} + \tau l_{t+1}$.

$l_t = l_0 - \eta\sum_{r=0}^{t-1}\nabla f(x_r)$ is the momentum term. The final update combines (small learning rate) gradient descent with this (large learning rate) momentum.

Eventually, for $\varepsilon > 0$, we need at most $\frac{\sqrt{2L}}{\sqrt{\varepsilon}}$ iterations to find a point x_{T_ε} with $f(T_\varepsilon) \leq \varepsilon$.

Recall: Gradient Descent needs $\frac{2L}{\varepsilon}$ iterations to find that point.

Lec 18 Interior Point Method

For a function f and a *convex* constraint set $\mathcal{D}$ with a ν-self-concordant barrier R,

Interior point method: at every step:

- Update: $\lambda_{t+1} = \lambda_t(1-\beta)$ for $\beta \in (0,1)$.
- Find the minimizer $x^*_{t+1} \approx \operatorname{argmin}\{f(x) + \lambda_{t+1}R(x)\}$ by running pre-conditioned gradient descent with learning rate η, starting from x^*_t

Observe: $\lambda_t \to 0$ as $t \to \infty$, so eventually we find an *approximate* minimizer of x in $\mathcal{D}$.

In other words: Self-concordant implies sandwich in Dikin's Ellipsoid

In other words, for ν-self-concordant barrier function R, to optimize $F_{t+1}(x) = \langle c, x\rangle + \lambda_{t+1}R(x)$ for $\lambda_{t+1} \in [(1-1/(16\sqrt{\nu}))\lambda_t, \lambda_t]$, the trajectory of pre-conditioned gradient descent (starting from x^*_t) stays inside the *nice region*: Dikin's Ellipsoid:

$$\mathcal{E}_t = \left\{x \in \mathbb{R}^d \mid (x-x^*_t)^\top \nabla^2 R(x^*_t)(x-x^*_t) \leq \frac{1}{8}\right\}$$

In particular, $x^*_{t+1} \in \mathcal{E}_t$.

In this region, for every $x \in \mathcal{E}_t$: $F_{t+1}(x)$ is a *sandwich function*:

$$\frac{1}{4}\nabla^2 R(x^*_t) \leq \nabla^2 F_{t+1}(x)/\lambda_{t+1} \leq 4\nabla^2 R(x^*_t)$$

1. It is only self-concordant barrier functions which look log-like, not self-concordant fun

2. Self-concordant functions must be convex functions. second inequality means "smoothness" implies convex.

3. Every nonempty, nondegenerate polytope in dimension d has a corresponding self-concordant function. A common approach to constructing a self-concordant function for a given polytope is to use the log barrier function.

I.Barrier Function

Intuition: transfer the constraint optimization $\min_{x\in\mathcal{D}} f(x)$ for a *convex set $\mathcal{D}$* to an unconstraint optimization problem by introducing the convex barrier function, a function of type

$$R(x) = \begin{cases} = +\infty & \text{if } x \in \partial\mathcal{D}; \\ \in (-\infty, +\infty) & \text{if } x \in \mathcal{D}^o. \end{cases}$$

Here, $\partial\mathcal{D}$ means the boundary of $\mathcal{D}$, $\mathcal{D}^o$ means the interior of $\mathcal{D}$

Minimize $f(x) + \lambda R(x)$ for a sufficiently small $\lambda > 0$ gives us *approximately* the minimizer of $f(x)$ in $\mathcal{D}$.

II.Self-concordant

A barrier function R for a *convex* set $\mathcal{D}$ in dimension d is called self-concordant with parameter ν, if for every $x \in \mathcal{D}^o$ and for every $v \in \mathbb{R}^d$:

$$\langle\nabla R(x), v\rangle^2 \leq \nu \times v^\top\nabla^2 R(x)v$$

$$|v^\top\nabla^2 R(x)v|^{3/2} \geq \frac{1}{2} \times \nabla^3 R(x)(v,v,v)$$

Here, $\nabla^3 R(x)(v,v,v) = \frac{d^3}{dt^3}R(x+tv)\,|_{t=0}$.

Actually, $\langle\nabla R(x), v\rangle = \frac{d}{dt}R(x+tv)\,|_{t=0}$, $v^\top\nabla^2 R(x)v = \frac{d^2}{dt^2}R(x+tv)$.

Why it is a $*\leq \nu*$? — We need the function to change slowly! The smaller ν is, the better.

Why it is a $*\geq 1/2*$? — Recall: We need the function to have some *curve*. The larger the Hessian, the more "curved" is the function.

Fact: Self-concordant barrier function is convex. (Implied by $\langle\nabla R(x), v\rangle^2 \leq \nu \times v^\top\nabla^2 R(x)v$).

Fact: If R_1, R_2 are ν self-concordant barrier function for set $\mathcal{D}$.

- Then $R_1 + R_2$ is 2ν self-concordant barrier function for set $\mathcal{D}$.
- Then $R_3(x) = R_1(Ax+b)$ is ν self-concordant for every matrix A ; vector b.

Fact: For every *convex* set $\mathcal{D}$ in $\mathbb{R}^d$, there is a self-concordant barrier function R for it with parameter d (the volumetric barrier function).

Lec 17 Newton and Precondition GD

Hessian gives more information about the shape of the function f around x.

$f(y) \leq f(x) + \langle\nabla f(x), y-x\rangle + \frac{1}{2}(y-x)^\top\nabla^2 f(x)(y-x) + O(||y-x||_2^3)$

1. Newton Method

At every iteration t, update (typically choose η=1): $y = x - \eta[\nabla^2 f(x)]^{-1}\nabla f(x)$ Computing $[\nabla^2 f(x_t)]^{-1}$is typically very coefficient. Newton's method only runs fast in certain special cases.

Newton's Method **only** has a local convergence guarantee. LCR of Netwon's Method is much **faster** than GD, since Netwon's method uses more finegrind local geometry information of the function: Hessian.

- Assuming x^* is a strict local minima of $f(x), \nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \geq \sigma I$: $||x_{t+1} - x^*||_2 \leq \frac{2L}{\sigma}||x_t - x^*||_2^2$

- Newton's method has a local **quadratic** convergence rate near a strict local minimum of the target function

2. Preconditioned GD (converge at a **linear rate** η/a)

A sandwich function is not a strongly convex function with good condition number.

There exists a positive definite matrix A and a value $a \geq 1$ such that for every x, $A \leq \nabla^2 f(x) \leq aA$

We have that for every $\eta \leq \frac{1}{a}$, the update $x_{t+1} = x_t - \eta[\nabla^2 f(x_t)]^{-1}\nabla f(x_t)$, converge in linear rate $||x_{t+1} - x^*||_2 \leq (1-\eta/a)||x_t - x^*||_2$- Newton's method has a global linear convergence rate on sandwich functions.

3. Adaptive algorithms finds precondition M automatically to significantly improve the convergence rate.

4. Newton's method is faster than GD for cvxopt in LCR, and only for special functions in GCR

5. Newton's method is useful for non-convex optimization. It will have a local convergence.

6. The convergence rate of Newton's method only depends on the L of the Hessian, NOT smooth or Lipschitz

7. L in the convergence radius ($|x_0 - x^*|$) was lipschitzness of the hessian, so the convergence radius of

8.Newton's method does NOT depend on the Smoothness or Lipschitzness of the function.

9.Newton's method need to calculate the Hessian, but precondition GD does not. Pre-condition only need to compute A^{-1}, for precondition to have a good convergence rate, it need to be a sandwich function but any matrix A can be considered a precondtioned GD

10.The performance of Newton's method greatly depends on the initial point, and its convergence can be sensitive to the local curvature of the function being optimized.

11. The convergence rate of pre-conditioned gradient descent on sandwich functions depends on the learning rate and a constant which is not related to the condition number

12. Preconditioned GD can have a global convergence rate on smooth convex functions. The preconditioning step in preconditioned gradient descent can help improve the conditioning of the optimization problem and accelerate the convergence rate. The global convergence rate depends on the smoothness and convexity properties of the function and the preconditioning matrix used.

Lec 19 Adaptive Algorithms

1. Gradient Sign Method

We can just scale down the gradient so each coordinate of the gradient has absolute value one.

Update $x_{t+1} = x_t - M_t^{-1}\nabla f(x_t)$ where $M_t = \text{diag}(|[\nabla f(x_t)]_1|, |[\nabla f(x_t)]_2|, \cdots, |[\nabla f(x_t)]_d|)$

If some coordinate of the function has bad smoothness, then gradient will be large at that coordinate, since its zig-zagging. If one coordinate of the gradient has been large for a while, scale it down, vise versa.

2. Adagrad: Adaptive algorithm that can find the "best scaling " of each coordinate automatically. Using the "sum of the past gradients" to define the scaling matrix, instead of just the last iteration gradient.

Prove to **converge in convex** setting.

3. Precondition vs Adagrad

Precondition GD: need to tune η carefully and choose the pre-conditioning matrix carefully.

Adagrad does NOT need to tune the η or choose the pre-conditioning matrix

4. Adam (Adagrad with Momentum) (Does NOT prove to converge, even may diverge in convex)

- At each iteration, computer Momentum $g_{t+1} = \gamma g_t + (1-\gamma)\nabla f(x_t)$ Compute scale factor $s^2_{t+1} = \beta s_t^2 + (1-\beta)[\nabla f(x_t)]^2$ Update the adaptive momentum: $x_{t+1} = x_t - \eta\text{diag}(s_{t+1})^{-1}g_{t+1}$

- But it works extremely well in practice for training neural networks.

5. Adam is faster than SGD with momentum because the direction of Adam is actually much lighter compared to the direction of SGD with momentum. It's not more robust to the choice of learning rate

6. **SGD is bad** in deep learning because of the existence of a small fraction of parameters with very bad (directional) smoothness.

7. Adagrad with stochastic gradient and a fixed learning rate **converges** to the optimal solution for smooth convex functions, as long as the **variance** of the stochastic gradient is **bounded**.

8. Sign of **SGD** with a fixed learning rate DOES NOT has a convergence guarantee for Smooth convex functions, as long as the variance of the stochastic gradient is bounded.

9. The use of a fixed learning rate in the gradient descent method does not provide a convergence guarantee for smooth convex functions. Although gradient descent can converge to the optimal solution for smooth convex functions, the fixed learning rate can lead to slow convergence or oscillations around the optimal solution, particularly if the learning rate is not chosen appropriately.

Lec 20 Distributed Optimization

1. Distributed Optimization: Each individual machine is assumed to have **infinite** computation power, but the **communication** between the machines are **limited**. The goal is to find the minimizer of the problem and minimize the communication between the machines.

2. Using GD, the per communication cost $O(md)$, where $W \in R^d$, m is the number of machines. The total number of iterations depends on the convergence rate of GD, and when the smoothness or the Lipschitzness of f is not very good, then the convergence rate GD is not very good.

3. But ADMM's convergence rate does **NOT** depend on smoothness or the Lipschitzness of f (proved using convex functions by Mirror Descent Analysis)

minimize W_j on each local machine, and together to obtain $W^{(t)}$instead of doing a gradient step on $W^{(t)}$

4. ADMM is a DATA-parallel technique. One of the main spirits behind ADMM is duality.

5. The per-iteration communication cost of ADMM and distbuted GD are the same.

6. ADMM convergence rate has nothing to do with data partition and only depends on number of machine

Lec 21 Online Optimization

How to minimize the regret:

$$R := \frac{1}{T}\sum_{t\in[T]} f_t(x_t) - \min_{x\in\mathcal{D}}\frac{1}{T}\sum_{t\in[T]} f_t(x)$$

When $f_t = f$ is a convex function, we know how to do it using gradient descent.

Key theory for today: When f_t's are convex functions, gradient descent (or mirror descent in general) also works!

Without assuming anything (such as f_t's are related or changing slowly etc).

The spirit of Mirror Descent: When $f_t(x_t)$ is much larger than $f_t(x)$, then x_t is moving closer to x.

Thus, $\nabla f_t(x_t)$ is "useful" to find the optimal point x^*, without any particular assumption on f_t (except convexity).

Gradient descent for online learning:

At each iteration t, update:

$$x_{t+1} = x_t - \eta\nabla f_t(x_t)$$

Theorem: when each f_t is convex and $L-$*Lipschitz*, let $x^* = \text{argmin}\sum_{t\in[T]} f_t(x)$. Using learning rate $\eta = \frac{\|x-x_0\|}{L\sqrt{T}}$, the regret is bounded by:

$$\frac{1}{T}\sum_{t\in[T]} f_t(x_t) - \frac{1}{T}\sum_{t\in[T]} f_t(x^*) \leq \frac{2\|x^* - x_0\|_2 L}{\sqrt{T}}$$

Why this is true? There might be no connection between each f_t.

Case 1: All f_t are sort of random, then there is no x such that $\sum_{t\in[T]} f_t(x)$ is small. So our baseline is pretty bad.

Case 2: Most of the f_t has a common minimizer x^*, then since f_t is convex, $-\nabla f_t(x_t)$ is pointing to x^*: The gradient information is useful!

We also take a special look at the Online Newton's Method, or called online BFGS optimization algorithm:

At each iteration, update using a quasi-newton step:

$$x_{t+1} = x_t - \eta_t B_t \nabla f_t(x_t)$$

Where the "inverse Hessian" matrix B_t is given by: for $s_{t+1} := \eta_t B_t \nabla f_t(x_t)$ and

$$y_t := \nabla f_t(x_t) - \nabla f_{t-1}(x_{t-1})$$

Update for $\rho_t = \langle s_t, y_t \rangle^{-1}$:

$$B_t = (I - \rho_t s_t y_t^\top) B_{t-1} (I - \rho_t y_t s_t^\top) + \rho_t s_t s_t^\top$$

The intuition of this update is we want to update B_t such that $B_t s_t \approx -y_t$.

Another major extension assumes that f_t is not changing very fast, in the sense that

$$V = \sum_{t=2}^{T} \|f_t - f_{t-1}\|$$

is bounded by a not so large value.

The regret can scale with $\frac{\sqrt{V}}{T}$ in some cases.

Stein's Method

At each time t, we can only get the function value $f_t(x_t)$ as the feedback, instead of $\nabla f_t(x_t)$.

Key technique: Using stokes formula: for every $r \geq 0$, for v being randomly sampled unit vector in $\mathbb{R}^d$ ($x \in \mathbb{R}^d$),

$$\mathbb{E}_v \left[\frac{d}{r} f(x + rv) v \right] = \nabla \tilde{f}(x)$$

Where $\tilde{f}(x) = \int_{\|v\|_2 \leq 1} f(x + rv) dv$

Thus, $\frac{d}{r} f(x + rv) v$ is a stochastic gradient for function $\tilde{f}(x)$.

Thus, we can query $x_t + rv$ and update

$$x_{t+1} = x_t - \frac{d}{r} f_t(x_t + rv) v$$

Extension

Another major extension want to complete with the best sequence of "slow moving" x_t^*, in the sense that the baseline now becomes:

In this setting, sometimes we can achieve competitive ratio $\text{poly}(d)$, in the sense that

Lec 22 Non-Convex Optimization

There are "saddle points", whose gradients are also zero but not "locally minimal".

In fact, in high dimension, one can construct a function where gradient descent almost always stuck at a saddle point.

For non-convex optimization: $\nabla^2 f(x)$ might not be positive semi-definite: This gives us local minimal and saddle points.

There are global minima, local minima, saddle points.

They are points whose gradients are zero, but have different properties.

A point x is called a local minima (second-order local minima), if $\nabla f(x) = 0$ and $\nabla^2 f(x)$ is PSD (positive semi-definite, i.e. $\nabla^2 f(x) \geq 0$).

In particular, the global minima ($x^* = \text{argmin} f(x)$) is a special local minima.

A point x is called a saddle point, if

$\nabla f(x) = 0$ and $\nabla^2 f(x)$ is not PSD – meaning there exists a $v \in \mathbb{R}^d$ such that $v^\top \nabla^2 f(x) v < 0$.

Hessian Descent

- We first introduce the most easily understandable non-convex optimization algorithm: The Hessian Descent:
- It is not used at all in practice, but it is still extremely important.
- It is the spirit behind all other algorithms to find a (second order) local minima.

Warning

Be very clear: when people talk about local minima, they typically mean second oder local minima.

They DO NOT mean a point x such that for every sufficiently small δ, $f(x+\delta) \geq f(x)$ – Such "local minima" CAN NOT BE FOUND EFFICIENTLY (there are many machine learning papers that switch definitions to cheat).

Recall we mentioned: Hessian Descent is not used at all in practice.

This is due to a simple reason: (noisy) Gradient Descent can already find a point whose gradient is close to zero, and whose Hessian is close to PSD efficiently.

In other words, (noisy) Gradient Descent can find an approximate (second-order) local minima efficiently.

Hessian Descent

Update (gradient descent): $y_{t+1} = x_t - \eta \nabla f(x_t)$.

Update (hessian descent): Find a unit vector v corresponding to the eigenvector of $\nabla^2 f(x_t)$ with the smallest (can be negative) eigenvalue, define

$$z_{t+1,1} = x_t - \eta v, \quad z_{t+1,2} = x_t + \eta v$$

Update

$$x_{t+1} = \text{argmin}\{f(y_{t+1}), f(z_{t+1,1}), f(z_{t+1,2})\}$$

Recall: For a L-smooth function, convex or not, using $\eta \leq \frac{1}{L}$, we have:

For the update $y_{t+1} = x_t - \eta \nabla f(x_t)$:

$$f(y_{t+1}) \leq f(x_t) - \frac{\eta}{2} \|\nabla f(x_t)\|_2^2$$

In other words, Gradient Large $\implies$ Large Objective Decrease.

Suppose $v^\top \nabla^2 f(x_t) v = -\delta$ for $\delta > 0$, for every $\eta \leq \frac{\delta}{4\gamma}$, we have

$$\frac{1}{2}\left(f(z_{t+1,1}) + f(z_{t+1,2})\right) \leq f(x_t) - \frac{\eta^2}{4}\delta$$

In other words, Hessian Negative (locally highly non-convex) $\implies$ Large Objective Decrease.

On the other hand, recall for the update $y_{t+1} = x_t - \eta \nabla f(x_t)$:

$$f(y_{t+1}) \leq f(x_t) - \frac{\eta}{2} \|\nabla f(x_t)\|_2^2$$

In other words, Gradient Large $\implies$ Large Objective Decrease.

Together, algorithm Hessian Descent can find a point whose gradient is close to zero, and whose Hessian is close to PSD efficiently.

Noisy Gradient Descent

Algorithm (noisy) Gradient Descent: At every iteration, update:

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

For every T_0 iterations, further update $x_{t+1} \leftarrow x_{t+1} + \xi_{t+1}$, where $\xi_{t+1} \sim \mathcal{N}(0, \sigma^2 I)$ is a small Gaussian Noise.

Gradient Descent might stuck at a saddle points, but adding a little bit noise can already help gradient descent algorithm efficiently escape saddle points.

In practice, since we are using stochastic gradient descent for most of the problems, adding such noise is usually unnecessary.

Now we are updating:

$$\nabla f(x_{t+1}) \approx (I - \eta \nabla^2 f(x_{T_1})) \nabla f(x_t)$$

This is the so-called power method, it will converge to the eigenvector of $(I - \eta \nabla^2 f(x_{T_1}))$ with the largest eigenvalue, as long as the starting point $\nabla f(x_{T_1})$ is not very adversarial, that is why we need to add a bit random noise.

Noisy Gradient Descent

In particular, when $\nabla^2 f(x_{T_1})$ is not PSD, then $\|(I - \eta \nabla^2 f(x_{T_1}))\|_2 > 1$, which means that gradient will become large again after some iterations, and gradient descent will decrease objective again.

The algorithm will stop only if (1). Gradient is small (2). Hessian is approximately PSD.

The rate of convergence for noisy gradient descent is $\text{poly}(1/\varepsilon, 1/\delta)$ to find a point whose gradient is smaller than ε and whose hessian is $\geq -\delta I$.

The rate is not very important, just the spirit is important: gradient descent with a little bit noise can already converge to an approximate (second order) local minima efficiently.

Lec 23 Bayesian Optimization

Bayesian optimization does not require computing the gradient of f, it only requires knowing the function value.

Bayesian optimization is a global optimization, it looks at the global structure of f. Gradient descent is local.

Bayesian optimization is a low dimension non-convex optimization algorithm, it is ineffective in high dimension. It is an analog of Ellipsoid algorithm, but for *non-convex* optimization.

Key applications: Obtaining higher quality solution comparing to gradient descent in low dimension optimization problem, and avoiding computing gradient (when computing the gradient is expensive). Such as hyper-parameter tuning, architecture search etc.

A function f_n is considered as "good" for a given set of observations $f(x_1), \cdots, f(x_n)$, if

When x is close to one of the x_i,

Then $f_n(x)$ is close to $f_n(x_i)$.

f_n is "smooth".

We will focus on a special type of Bayesian optimization: The Gaussian Process Regression.

Given a kernel function $K(x, y) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$:

Recall: A kernel function is a function satisfies for any set of points $x_1, \cdots, x_t$, the $t \times t$ matrix M defined as:

$$M_{i,j} = K(x_i, x_j)$$

is PSD.

Given a kernel function $K(x, y) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$:

Define

$$v_t(x) = (K(x, x_0), K(x, x_1), \cdots, K(x, x_t))^\top, F_t = (f(x_0), f(x_1), \cdots, f(x_t))^\top$$

Define matrix $M_t : \mathbb{R}^{(t+1)\times(t+1)}$ such that

$$[M_t]_{i+1,j+1} = K(x_i, x_j)$$

Define $P_t(x)$ as a Gaussian distribution:

$$P_t(x) \sim \mathcal{N}(\mu_t(x), \sigma_t^2(x))$$

Where

$$\mu_t(x) = v_t(x)^\top M_t^{-1} F_t$$

Where

$$\sigma_t^2(x) = K(x, x) - v_t(x)^\top M_t^{-1} v_t(x)$$

μ_t, σ_t^2 are "smooth functions", μ_t defines the "mean", σ_t^2 defines the "confidence interval".

Recall the goal: Find an approximate maximizer of f.

Key idea: Query the point x_{t+1} such that $f_t(x_{t+1})$ is the largest in expectation, for $f_t \sim P_t$.

Define $f_t^* = \max_{j<0} \{f(x_j)\}$, we will find the point x_{t+1} such that

$$x_{t+1} = \text{argmax}_x \mathbb{E}_{g \sim P_t} [[g(x) - f_t^*]^+]$$

Here $[z]^+ = \text{ReLU}(z)$.

Here, $\text{EI}_t(x) = \mathbb{E}_{g \sim P_t}[[g(x) - f_t^*]^+]$ is called the acquisition function

Using $[g(x) - f_t^*]^+$ is often referred to as an "optimistic exploration"

If $\mu(x)$ is smaller than $\mu(x')$, but $\sigma^2(x)$ is much larger than $\sigma^2(x')$, then the algorithm would prefer x.

The theory for rate of convergence is in general unknown, this is a heriustic algorithm.

Lec 24 Overparameterization

Principle of over-parameterization: by making the number of parameters in the optimization problem much larger than necessary, it makes the underlying optimization problem easier.

Intuitively, we change the optimization problem from $\min_{w \in \mathbb{R}^d} f(w)$ to $\min_{w \in \mathbb{R}^D} f(w)$, where $D \gg d$.

Intuitively, we will have much more directions to search through when optimizing f, and less likely to stuck at a local minima.

Lec 24 Overparameterization

Here, easier typically means that the new optimization has less or no bad local minima.

Indeed, for the larger model, the per-iteration optimization cost is higher, but the quality of the solution can also be much higher.

Intuitively, when we have a larger model, we have more degrees of freedom when searching through the parameter space, and less likely to stuck at a bad local minima.

Lec 25 Sampling based Optimization

Simulated Annealing: Starting from distribution $p(x) \propto e^{-0 \times f(x)}$, we gradually increase the value of λ to $+\infty$ and sample from $p_\lambda(x) \propto e^{-\lambda f(x)}$ at every iteration.

Intuition: When λ_1 is close to λ_2, sampling from $p_{\lambda_2}(x)$ using uniform samples from $p_{\lambda_1}(x)$ as a warm start is easier.

The Metropolis–Hastings algorithm to sample from a distribution $P(x)$ given samples from distribution $Q(x)$ as a warm start:

Initially, sample a point x_0 from distribution $Q(x)$.

At every iteration t, sample $y_{t+1} \sim \mathcal{N}(x_t, \sigma^2 I)$ for some $\sigma > 0$ (to be tuned based on your specific application).

Define $\alpha = \min\left\{\frac{P(y_{t+1})}{P(x_t)}, 1\right\}$

Define

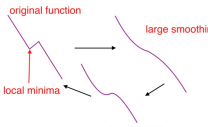
$$x_{t+1} = \begin{cases} y_{t+1} & \text{w.p. } \alpha \\ x_t & \text{w.p. } 1 - \alpha. \end{cases}$$

The spirit of simulated annealing algorithm to minimize a function f: Evolve the distribution p from uniform at random to supports only on the minimizers of f.

When we run Metropolis–Hastings algorithm for sufficiently many steps at every iteration (steps $\to +\infty$), simulated annealing converges to the global minimizer of f.

However, the number of steps might be very large, there is no efficient rate guarantee unless f is convex.

ES can even escape local minima when σ_t is large.



Main theorem: When $t \to \infty$, $x_t \to$ a sample according to distribution $P(x)$.

In other words, the stationary distribution of the Metropolis–Hastings process is $P(x)$.

Let $n(x \mid x')$ be the probability density of point x associated with distribution $\mathcal{N}(x', \sigma^2 I)$.

Key idea: using a warm start Q that is close to P, Metropolis–Hastings algorithm converges to the distribution P faster.

This is also heuristics (but there is a very non-trivial convergence proof when P, Q are log-concave distributions).

Evolutionary strategies to maximize a function $f : \mathbb{R}^d \to \mathbb{R}$

At every iteration t, randomly sample N i.i.d. vectors $\epsilon_1, \cdots, \epsilon_N \sim \mathcal{N}(0, I_{d \times d})$.

Compute function value using a standard deviation σ_t: $f_i = f(x_t + \sigma_t \epsilon_i)$ for every $i \in [N]$.

Update using learning rate η:

$$x_{t+1} = x_t + \eta \frac{1}{N\sigma_t} \sum_{i=1}^{N} f_i \epsilon_i$$

The "lucky ones" ϵ_i with higher function values f_i gets weighted higher.

Actually, the ES algorithm is trying to do stochastic gradient ascent on the new objective

$$F_t = f * g_t$$

Where g_t is the density function of $\mathcal{N}(0, \sigma_t^2 I)$, $*$ is the convolution operation:

$$[f * g](x) = \int_y f(y) g(y - x) dy$$

By Stoke's formula,

$$\nabla F_t = f * \nabla g_t$$

There are a lot of other evolutionary algorithms.

The spirit: Initially, starting from a prior distribution over the set of parameters.

At every iteration, evolve this distribution a little bit towards higher quality solutions

T Gradient Descent with a proper learning rate will converge to a local minimum when optimizing over smooth non-convex functions.

T Gradient Descent with a properly selected learning rate will converge to a critical minimum when optimizing over any functions, convex or not

T Hessian Descent with a properly selected learning rate will converge to an approximate 2-order local minimum for any smooth nonconvex functions.

F Hessian Descent is a widely used optimization algorithm in practice for non-convex optimization.

F Hessian Descent is a first-order optimization algorithm.

T Noisy gradient descent is a first-order optimization algorithm.

F Bayesian Optimization is very effective in high-dimension non-convex optimization.

T One of the main applications of Bayesian Optimization is hyper-parameter search in deep learning.

F One of the main applications of Bayesian Optimization is training the weights of the neural network.

F Bayesian Optimization is a first-order optimization algorithm.

F Bayesian Optimization requires the target function to be smooth and Lipschitz.

T Bayesian Optimization uses optimistic exploration

T A global minimum is a local minimum.

F An infinite order local minimum is a global minimum.

F An infinite order local minimum is a saddle point.

T An infinite order local minimum is a second order local minimum.

F A second-order local minimum is an infinite-order local minimum.

F Finding an approximate infinite order local minimum can be done efficiently, as long as the function is analytic.

T A Saddle point must have zero gradients.

T Mirror Descent with a proper learning rate can converge to an (approximate) critical point for smooth and Lipschitz non-convex functions.