Princeton Univ. F'20 COS 521: Advanced Algorithm Design

Lecture 9: Going with the slope: offline, online, and randomly

Lecturer: *Sahil Singla* Last updated: September 28, 2020

This lecture is about *gradient descent*, a popular method for continuous optimization (especially nonlinear optimization).

We start by recalling that allowing nonlinear constraints in optimization leads to NP-hard problems in general. For instance the following single constraint can be used to force all variables to be 0/1.

$$\sum_i x_i^2(1-x_i)^2 = 0.$$

Notice, this constraint is nonconvex. We'll see in the next lecture that the Ellipsoid method can solve *convex* optimization problems efficiently under fairly general conditions. But it is slow in practice.

Gradient descent is a popular alternative because it is simple and it gives some kind of meaningful result for both *convex* and *nonconvex* optimization. It tries to improve the function value by moving in a direction related to the *gradient* (i.e., the first derivative). For convex optimization it gives the global optimum under fairly general conditions. For nonconvex optimization it arrives at a local optimum.

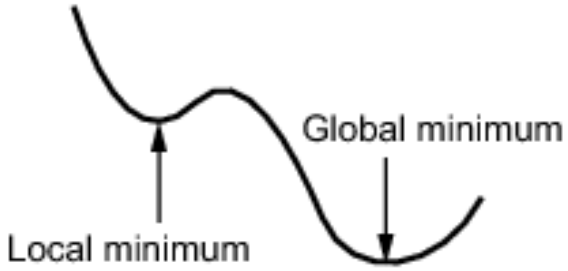

Figure 1: For nonconvex functions, a local optimum may be different from the global optimum

We will first study unconstrained gradient descent where we are simply optimizing a function $f(\cdot)$. Recall that the function is *convex* if $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ for all x, y and $\lambda \in [0,1]$.

1 Gradient descent for convex functions: univariate case

The gradient for a univariate function f is simply the derivative: $f'(x)$. If this is negative, the value decreases if we increase x a little, and increases if we decrease f. Gradient descent consists of evaluating the derivative and moving a small amount to the right (i.e., increase

x) if $f'(x) < 0$ and to move to the left otherwise. Thus the basic iteration is $x \leftarrow x - \eta f'(x)$ for a tiny η called *step size.*

The function is *convex* if between every two points x, y the graph of the function lies *below* the line joining $(x, f(x))$ and $(y, f(y))$. It need not be differentiable everywhere but when all derivatives exist we can do the *Taylor expansion*:

$$f(x+\eta) = f(x) + \eta f'(x) + \frac{\eta^2}{2} f''(x) + \frac{\eta^3}{3!} f'''(x) \cdots . \tag{1}$$

If $f''(x) \geq 0$ for all x then the the function is *convex*. This is because $f'(x)$ is an *increasing* function of x. The minimum is attained for x where $f'(x) = 0$ since $f'(x)$ is +ve to the right of it and −ve to the left. Thus moving both left and right of this point increases f and it never drops. The function is *concave* if $f''(x) \leq 0$ for all x; such functions have a unique maximum.

Examples of convex functions: $ax + b$ for any $a, b \in \Re$; $\exp(ax)$ for any $a \in \Re$; x^α for $x \geq 0$, $\alpha \geq 1$ or $\alpha \leq 0$. Another interesting example is the negative entropy: $x \log x$ for $x \geq 0$.

Examples of concave functions: $ax + b$ for any $a, b \in \Re$; x^α for $\alpha \in [0, 1]$ and $x \geq 0$; $\log x$ for $x \geq 0$.

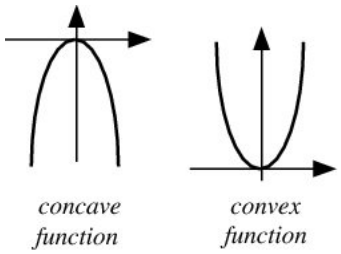

Figure 2: Concave and Convex Function

To minimize a convex function by gradient descent we start at some x_0 and at step i update x_i to $x_{i+1} = x_i + \eta f'(x)$ for some small $\eta < 0$. In other words, move in the direction where f *decreases*. If we ignore terms that involve η^3 or higher, then

$$f(x_{i+1}) = f(x_i) + \eta f'(x_i) + \frac{\eta^2}{2} f''(x_i).$$

and the best value for η (which gives the most reduction in one step) is $\eta = -f'(x)/2f''(x)$, which gives

$$f(x_{i+1}) = f(x_i) - \frac{(f'(x_i))^2}{2f''(x_i)}.$$

Thus the algorithm makes progress so long as $f''(x_i) > 0$. Convex functions that satisfy $f''(x) > 0$ for all x are called *strongly convex*.

The above calculation is the main idea in *Newton's method*, which you may have seen in calculus. Proving convergence requires further assumptions.

2 Convex multivariate functions

A convex function on $\Re^n$, if it is differentiable, satisfies the following basic inequality, which says that the function lies "above "the tangent plane at any point.

$$f(x+z) \geq f(x) + \nabla f(x) \cdot z \quad \forall x, z. \tag{2}$$

Here $\nabla f(x)$ is the vector of first order derivatives where the ith coordinate is $\partial f/\partial x_i$ and called the *gradient*. Sometimes we restate it equivalently as

$$f(x) - f(y) \leq \nabla f(x) \cdot (x-y) \quad \forall x, y \tag{3}$$

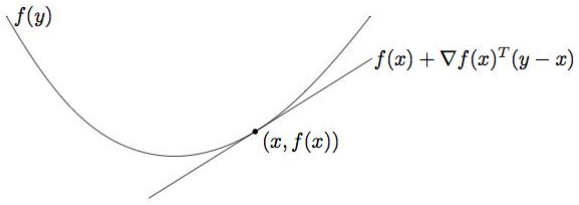

Figure 3: A differentiable convex function lies above the tangent plane $f(x)+\nabla f(x)\cdot(y-x)$

If higher derivatives also exist, the multivariate Taylor expansion for an n-variate function f is

$$f(x+y) = f(x) + \nabla f(x) \cdot y + y^T \nabla^2 f(x) y + \cdots . \tag{4}$$

Here $\nabla^2 f(x)$ denotes the $n \times n$ matrix whose i, j entry is $\partial^2 f/\partial x_i \partial x_j$ and it is called the *Hessian*. It can be checked that f is *convex* if the Hessian is positive semidefinite; this

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Figure 4: The Hessian

means $y^T \nabla^2 f y \geq 0$ for all y.

Example 1. *The following are some examples of convex functions.*

- Norms. *Every ℓ_p norm is convex on $\Re^n$. The reason is that a norm satisfies triangle inequality: $|x+y| \leq |x| + |y|$ $\forall x, y$.*

- $f(x) = \log(e^{x_1} + e^{x_2} + \cdots + e^{x_n})$ *is convex on* $\Re^n$. *This fact is used in practice as an analytic approximation of the* max *function since*

$$\max\{x_1, \ldots, x_n\} \leq f(x) + \leq \max\{x_1, \ldots, x_n\} + \log n.$$

  Turns out this fact is at the root of the multiplicative weight update method; the algorithm for approximately solving LPs that we saw in HW2 can be seen as doing a gradient descent on this function, where the x_i*'s are the* slacks *of the linear constraints. (For a linear constraint* $a^T z \geq b$ *the slack is* $a^T z - b$.*)*

- $f(x) = x^T Ax = \sum_{ij} A_{ij} x_i x_j$ *where* A *is positive semidefinite. Its Hessian is* A.

Some important examples of concave functions are: geometric mean $(\prod_{i=1}^n x_i)^{1/n}$ *and* log-determinant *(defined for* $X \in \Re^{n^2}$ *as* $\log\det(X)$ *where* X *is interpreted as an* $n \times n$ *matrix).*

Many famous inequalities in mathematics (such as Cauchy-Schwartz) are derived using convex functions. □

Example 2 (Linear equations with PSD constraint matrix)**.** *In linear algebra you learnt that the method of choice to solve systems of equations* $Ax = b$ *is Gaussian elimination. In many practical settings its* $O(n^3)$ *running time may be too high. Instead one does gradient descent on the function* $\frac{1}{2}x^T Ax - b^T x$*, whose local optimum satisfies* $Ax = b$ *(if* A *is symmetric).*[1] *If* A *is positive semidefinite this function is also convex since the Hessian is* A*, and gradient descent will actually find the solution. (Actually in real life these are optimized using more advanced methods such as* conjugate gradient*.) Also, if* A *is* diagonal dominant*, a stronger constraint than PSD, then Spielman and Teng (2003) have shown how to solve this problem in time that is* near linear *in the number of nonzero entries. This has had surprising applications to basic algorithmic problems like max-flow.*

Example 3 (Least squares)**.** *In some settings we are given a set of points* $a_1, a_2, \ldots, a_m \in \Re^n$ *and some* data values $b_1, b_2, \ldots, b_m$ *taken at these points by some function of interest. We suspect that the unknown function is a* line*, except the data values have a little error in them. One standard technique is to find a* least squares *fit: a line that minimizes the sum of squares of the distance to the datapoints to the line. The objective function is* $\min |Ax - b|_2^2$ *where* $A \in \Re^{m \times n}$ *is the matrix whose rows are the* a_i*'s. (The solution is also the first singular vector.) This objective is just* $x^T A^T Ax - 2(Ax)^T b + b^T b$*, which is convex.*

In the univariate case, gradient descent has a choice of only two directions to move in: *right* or *left*. In n dimensions, it can move in any direction in $\Re^n$. The most direct analog of the univariate method is to move diametrically opposite from the *gradient*.

The most direct analogue of our univariate analysis would be to assume a *lowerbound* of $y^T \nabla^2 f y$ for all y (in other words, a lowerbound on the eigenvalues of $\nabla^2 f$). In the rest of lecture we will only assume (2).

[1] To see this, expand the function out as $\sum_{i,j} x_i x_j A_{ij}/2 - \sum_i b_i x_i = \sum_i x_i(-b_i + x_i A_{ii}/2 + \sum_{j \neq i} A_{ij} x_j)$. So the partial with respect to x_i is: $-b_i + A_{ii} x_i + \sum_{j \neq i} A_{ij} x_j$, which equals zero if and only if $\langle A_i, x \rangle = b_i$. So all local optima satisfy $Ax = b$.

3 Gradient Descent for Constrained Optimization

As studied in previous lectures, constrained optimization consists of solving the following where $\mathcal{K}$ is a convex set and $f(\cdot)$ is a convex function.

$$\min f(x) \quad \text{s.t.} \quad x \in \mathcal{K}.$$

Example 4 (Spam classification via SVMs)**.** *This example will run through the entire lecture. Support Vector Machine is the name in machine learning for an algorithm to learn* linear classifier*; we saw these before in Lecture 6 (Linear Thinking). Suppose we wish to train the classifier to classify emails as spam/nonspam. Each email is represented using a vector in $\Re^n$ that gives the frequencies of various words in it ("bag of words"model). Say $a_1, a_2, \ldots, a_N$ are the emails, and for each there is a corresponding bit $b_i \in \{-1, 1\}$ where $b_i = 1$ means X_i is spam. SVMs use a* linear classifier *to separate spam from nonspam. If spam were perfectly identifiable by a linear classfier, there would be a function $W \cdot x$ such that $W \cdot a_i \geq 1$ if a_i is spam, and $W \cdot a_i \leq -1$ if not. In other words,*

$$1 - b_i W \cdot a_i \leq 0 \quad \forall i \tag{5}$$

Of course, in practice a linear classifier makes errors, so we have to allow for the possibility that (5) is violated by some a_i's. The obvious thing to try is to find a W that satisfies as many of the constraints as possible, but that leads to a nonconvex NP-hard problem. (Even approximating this weakly is NP-hard.) Thus a more robust version of this problem is

$$\min \sum_i \mathit{Loss}(1 - W \cdot (b_i a_i)) \tag{6}$$
$$|W|_2^2 \leq n \qquad \textit{(scaling constraint)}$$

where Loss$(\cdot)$ is a function that penalizes unsatisfied constraints according to the amount by which they are unsatisfied. (Note that W is the vector of variables, and the scaling constraint gives meaning to the separation of "1 "in (5) by saying that W is a vector in the sphere of radius n, which is a convex constraint.) The most obvious loss function would be to count the number of unsatisfied constraints *but that is nonconvex. For this lecture we focus on convex loss functions; the simplest is the* hinge loss*: Loss$(t) = \max\{0, t\}$. Applying it to $1 - W \cdot (b_i a_i)$ insures that correctly classified emails contribute 0 to the loss, and incorrectly classified emails contribute as much to the loss as the amount by which they fail the inequality. The function in (6) is convex because the function inside Loss() is linear and thus convex, and Loss() preserves convexity since it can only lift the value of the linear function even further.*

If $x \in \mathcal{K}$ is the current point and we use the gradient to step to $x - \eta \triangle x$ then in general this new point will not be in $\mathcal{K}$. Thus one needs to do a *projection.*

Definition 1. *The* projection *of a point y on $\mathcal{K}$ is $x \in \mathcal{K}$ that minimizes $|y - x|_2$. (It is also possible to use other norms than ℓ_2 to define projections.)*

A projection oracle *for the convex body a black box that, for every point y, returns its projection on $\mathcal{K}$.*

Often convex sets used in applications are simple to project to.

Example 5. *If $\mathcal{K}$ = unit sphere, then the projection of y is $y/|y|_2$.*

Here is a simple algorithm for solving the constrained optimization problem. The algorithm only needs to access f via a *gradient oracle* and $\mathcal{K}$ via a *projection oracle*.

Definition 2 (Gradient Oracle)**.** *A* gradient oracle *for a function f is a black box that, for every point z, returns $\nabla f(z)$ the gradient valuated at point z. (Notice, this is a linear function of the form gx where g is the vector of partial derivatives evaluated at z.)*

The same value of η will be used throughout.

Gradient Descent for Constrained Optimization
Let $\eta = \frac{D}{G\sqrt{T}}$.
Repeat for $i = 0$ **to** T
$y^{(i+1)} \leftarrow x^{(i)} - \eta \nabla f(x^{(i)})$
$x^{(i+1)} \leftarrow$ Projection of $y^{(i+1)}$ on $\mathcal{K}$.
At the end output $z = \frac{1}{T}\sum_i x^{(i)}$.

Let us analyse this algorithm as follows. Let x^* be the point where the optimum is attained. Let G denote an upperbound on $|\nabla f(x)|_2$ for any $x \in \mathcal{K}$, and let $D = \max_{x,y\in\mathcal{K}} |x-y|_2$ be the so-called *diameter* of $\mathcal{K}$. To ensure that the output z satisfies $f(z) \leq f(x^*) + \epsilon$ we will use $T = \frac{4D^2G^2}{\epsilon^2}$.

Since $x^{(i)}$ is a projection of $y^{(i)}$ on $\mathcal{K}$, we have $(x^* - x^{(i+1)}) \cdot (y^{(i+1)} - x^{(i+1)}) \leq 0$, i.e., the vectors form an obtuse angle (can be formally proved using the Separating Hyperplane theorem). This gives

$$\begin{aligned}\left|x^{(i+1)} - x^*\right|^2 &\leq \left|y^{(i+1)} - x^*\right|^2 \\ &= \left|x^{(i)} - x^* - \eta\nabla f(x^{(i)})\right|^2 \\ &= \left|x^{(i)} - x^*\right|^2 + \eta^2\left|\nabla(f)(x^{(i)})\right|^2 - 2\eta\nabla f(x^{(i)}) \cdot (x^{(i)} - x^*).\end{aligned}$$

Reorganizing and using definition of G we obtain:

$$\nabla f(x^{(i)}) \cdot (x^* - x^{(i)}) \leq \frac{1}{2\eta}(\left|x^{(i)} - x^*\right|^2 - \left|x^{(i+1)} - x^*\right|^2) + \frac{\eta}{2}G^2.$$

Using (3), we can lowerbound the left hand side by $f(x^{(i)}) - f(x^*)$. We conclude that

$$f(x^{(i)}) - f(x^*) \leq \frac{1}{2\eta}(\left|x^{(i)} - x^*\right|^2 - \left|x^{(i+1)} - x^*\right|^2) + \frac{\eta}{2}G^2. \tag{7}$$

Now sum the previous inequality over $i = 1, 2, \ldots, T$ and use the telescoping cancellations to obtain

$$\sum_{i=1}^{T}(f(x^{(i)}) - f(x^*)) \leq \frac{1}{2\eta}(\left|x^{(0)} - x^*\right|^2 - \left|x^{(T)} - x^*\right|^2) + \frac{T\eta}{2}\left|G\right|^2.$$

Finally, by convexity $f(\frac{1}{T}\sum_i x^{(i)}) \leq \frac{1}{T}\sum_i f(x^{(i)})$ so we conclude that the point $z = \frac{1}{T}\sum_i x^{(i)}$ satisfies

$$f(z) - f(z^*) \leq \frac{D^2}{2\eta T} + \frac{\eta}{2}G^2.$$

Now set $\eta = \frac{D}{G\sqrt{T}}$ to get an upperbound on the right hand side of $2\frac{DG}{\sqrt{T}}$. Since $T = \frac{4D^2G^2}{\epsilon^2}$ we see that $f(z) \leq f(x^*) + \epsilon$.

4 Online Gradient Descent

In online gradient descent we deal with the following scenario. There is a convex set $\mathcal{K}$ given via a projection oracle. For $i = 1, 2, \ldots, T$ we are presented at step i a convex function f_i. At step i we have to put forth our *guess* solution $x^{(i)} \in \mathcal{K}$ but the catch is that we do not know the functions that will be presented in future. So our online decisions have to be made such that if x^* is the point w that minimizes $\sum_i f_i(w)$ (i.e. the point that we would have chosen in *hindsight* after all the functions were revealed) then the following quantity (called *regret*) should stay small:

$$\sum_i f_i(x^{(i)}) - f_i(x^*).$$

This notion should remind you of multiplicative weights, except here we may have general convex functions as "payoffs."

Example 6 (Spam classification against adaptive adversaries)**.** *We return to the spam classification problem of Example 4, with the new twist that this classifier* changes *over time, as spammers learn to evade the current classifier. Thus there is no* fixed *distribution of spam emails and it is fruitless to train the classifier at one go. It is better to have it improve and adapt itself as new emails arrive. At step t the optimum classifier f_t may not be known and is presented using a gradient oracle. This function just corresponds to the term in (6) corresponding to the latest email that was classfied as spam/nonspam. The goal is to do as well as the best single classfier we would want to use in hindsight.*

Zinkevich noticed that the analysis of gradient descent applies to this much more general scenario. Specifically, modify the above gradient descent algoritheorem to this problem by replacing $\nabla f(x^{(i)})$ by $\nabla f_i(x^{(i)})$. This algorithm is called *Online Gradient Descent*. The earlier analysis works essentially unchanged, once we realize that the left hand side of (7) has the regret for trial i. Summing over i gives the total regret on the left side, and the right hand side is analysed and upperbounded as before. Thus we have shown:

Theorem 1 (Zinkevich 2003)**.** *If D is the diameter of K and G is an upperbound on the norm of the gradient of any of the presented functions, and η is set to $\frac{D}{G\sqrt{T}}$ then the regret per step after T steps is at most $\frac{2DG}{\sqrt{T}}$.*

4.1 Case Study: Online Shortest Paths

The Online Shortest Paths problem models a commuter trying to find the best path with fewest traffic delays. The traffic pattern changes from day to day, and she wishes to have the smallest average delay over many days of experimentation.

We are given a graph $G = (V, E)$ and two nodes s, t. At each time period i, the decision maker selects one path p_i from the set $P_{s,t}$ of all paths that connect s, t (the choice for the day's commute). Then, an adversary independently chooses a weight function $w_i : E \to \mathbb{R}$ (the traffic delays). The decision maker incurs a loss equal to the weight of the path he or she chose: $\sum_{e \in p_i} w_i(e)$.

The problem of finding the best would be natural to consider this problem in the context of expert advice. We could think of every element of $P_{s,t}$ as an expert and apply the multiplicative weights algorithm we have seen before. There is one major flaw with this approach: there may be exponentially many paths connecting s, t in terms of the number of nodes in the graph. So the updates take exponential time and space in each step, and furthermore the algorithm needs too long to converge to the best solution.

Online gradient descent can solve this problem, once we realize that we can describe the set of all distributions x over paths $P_{s,t}$ as a convex set $\mathcal{K} \in \mathbb{R}^m$, with $O(|E| + |V|)$ constraints. Then the decision maker's expected loss function would be $f_i(x) = w_i^T \cdot x$. The following formulation of the problem as a convex polytope allows for efficient algorithms with provable regret bounds.

$$\sum_{e=(s,w), w \in V} x_e = \sum_{e=(w,t), w \in V} x_e = 1 \qquad \text{Flow value is 1.}$$

$$\forall w \in V, w \neq u, v, \sum_{e \ni w} x_e = 0 \qquad \text{Flow conservation.}$$

$$\forall e \in E, 0 \leq x_e \leq 1 \qquad \text{Capacity constraints.}$$

What is the meaning of the decision maker's move being a distribution over paths? It just means a fractional solution. This can be decomposed into a combination of paths as in the lecture on approximation algorithms. She picks a random path from this distribution; the expected regret is unchanged.

4.2 Case Study: Portfolio Management

Let's return to the portfolio management problem discussed in context of multiplicative weights. We are trying to invest in a set of n stocks and maximize our wealth. For $t = 1, 2, \ldots$, let $r^{(t)}$ be the vector of relative price increase on day t, in other words

$$r_i^{(t)} = \frac{\text{Price of stock } i \text{ on day } t}{\text{Price of stock } i \text{ on day } t-1}.$$

Some thought shows (confirming conventional wisdom) that it can be very suboptimal to put all money in a single stock. A strategy that works better in practice is *Constant Rebalanced Portfolio* (CRB): decide upon a *fixed* proportion of money to put into each stock, and buy/sell individual stocks each day to maintain this proportion.

Example 1. *Say there are only two assets,* stocks *and* bonds. *One CRB strategy is to put split money equally between these two. Notice what this implies: if an asset's price falls, you tend to* buy more of it*, and if the price rises, you tend to* sell it*. Thus this strategy roughly implements the age-old advice to "buy low, sell high."Concretely, suppose the prices each day fluctuate as follows.*

	Stock $r^{(t)}$	Bond $r^{(t)}$
Day 1	4/3	3/4
Day 2	3/4	4/3
Day 3	4/3	3/4
Day 4	3/4	4/3
...	...	...

Note that the prices go up and down by the same ratio on alternate days, so money parked fully in stocks or fully in bonds earns nothing in the long run. (Aside: This kind of fluctuation is not unusual; it is generally observed that bonds and stocks move in opposite directions.) And what happens if you split your money equally between these two assets? Each day it increases by a factor $0.5 \times (4/3 + 3/4) = 0.5 \times 25/12 \approx 1.04$*. Thus your money grows exponentially!*

Exercise: *Modify the price increases in the above example so that keeping all money in stocks or bonds alone will cause it to drop exponentially, but the 50-50 CRB increases money at an exponential rate.*

CRB uses a fixed split among n assets, but what is this split? Wouldn't it be great to have an angel whisper in our ears on day 1 what this magic split is? Online optimization is precisely such an angel. Suppose the algorithm uses the vector $x^{(t)}$ at time t; the ith coordinate gives the proportion of money in stock i at the start of the tth day. Then the algorithm's wealth increases on t by a factor $r^{(t)} \cdot x^{(t)}$. Thus the goal is to find $x^{(t)}$'s to maximize the final wealth, which is

$$\prod_t r^{(t)} \cdot x^{(t)}.$$

Taking logs, this becomes

$$\sum_t \log(r^{(t)} \cdot x^{(t)}) \tag{8}$$

For any fixed $r^{(1)}, r^{(2)}, \ldots$ this function happens to be concave, but that is fine since we are interested in maximization. Now we can try to run online gradient descent on this objective. By Zinkevich's theorem, the quantity in (8) converges to

$$\sum_t \log(r^{(t)} \cdot x^*) \tag{9}$$

where x^* is the best money allocation in hindsight.

This analysis needs to assume very little about the $r^{(t)}$'s, except a bound on the norm of the gradient at each step, which translates into a weak condition on price movements. In the next homework you will apply this simple algorithm on real stock data.

5 Stochastic Gradient Descent

(This was not covered in class.)

Stochastic gradient descent is a variant of the algorithm in Section 3 that works with convex functions presented using an even weaker notion: an *expected gradient* oracle. Given a point z, this oracle returns a linear function $gx + f$ that is drawn from a probability distribution $\mathcal{D}_z$ such that the expectation $E_{g,f\in\mathcal{D}_z}[gx + f]$ is exactly the gradient of f at z.

Example 7 (Spam classification using SGD)**.** *Returning to the spam classification problem of Example 4, we see that the function in (6) is a sum of many similar terms. If we randomly pick a single term and compute just its gradient (which is very quick to do!) then by linearity of expectations, the expectation of this gradient is just the true gradient. Thus the expected gradient oracle may be a much faster computation than the gradient oracle (a million times faster if the number of email examples is a million!). In fact this setting is not atypical; often the convex function of interest is a sum of many similar terms.*

Stochastic gradient descent can be analysed using *Online Gradient Descent* (OGD). Let $g_i \cdot x$ be the gradient at step i. Then we use this function —which is a linear function and hence convex— as f_i in the ith step of OGD. Let $z = \frac{1}{T}\sum_{i=1}^{T} x^{(i)}$. Let x^* be the point in $\mathcal{K}$ where f attains its minimum value.

Theorem 2. $\mathbb{E}[f(z)] \leq f(x^*) + \frac{2DG}{\sqrt{T}}$, *where D is the diameter as before and G is an upperbound of the norm of any gradient vector ever output by the oracle.*

Proof.

$$\begin{aligned}
\mathbb{E}[f(z) - f(x^*)] &\leq \frac{1}{T}\mathbb{E}[\sum_i (f(x^{(i)}) - f(x^*))] && \text{by convexity of } f \\
&\leq \frac{1}{T}\sum_i \mathbb{E}[\nabla f(x^{(i)}) \cdot (x^{(i)} - x^*)] && \text{using (2)} \\
&= \frac{1}{T}\sum_i \mathbb{E}[g_i \cdot (x^{(i)} - x^*)] && \text{Since expected gradient is the true gradient} \\
&= \frac{1}{T}\sum_i \mathbb{E}[f_i(x^{(i)}) - f_i(x^*)] && \text{Defn. of } f_i \\
&= \frac{1}{T}\mathbb{E}[\sum_i (f_i(x^{(i)}) - f_i(x^*)]
\end{aligned}$$

and the theorem now follows since the expression in the $\mathbb{E}[\cdot]$ is just the regret, which is *always* upperbounded by the quantity given in Zinkevich's theorem, so the same upperbound holds also for the expectation. □

6 Hints of more advanced ideas

Gradient descent algorithms come in dozens of flavors. (The Boyd-Vandenberghe book is a good resource, and Nesterov's lecture notes are terser but still have a lot of intuition.)

We know the optimal running time (i.e., number of iterations) of gradient descent in the oracle model; see the books by Hazan and Bubeck.

Surprisingly, just going along the gradient (more precisely, diametrically opposite direction from gradient) is not always the best strategy. *Steepest descent* direction is defined by quantifying the best decrease in the objective function obtainable via a step of unit length. The catch is that different norms can be used to define "unit length."For example, if distance is measured using ℓ_1 norm, then the best reduction happens by picking the largest coordinate of the gradient vector and reducing the corresponding coordinate in x (*coordinate descent*). The classical *Newton method* is a subcase where distance is measured using the *ellipsoidal norm* defined using the *Hessian*.

Gradient descent ideas underlie recent advances in algorithms for problems like Spielman-Teng style solver for Laplacian systems, near-linear time approximation algorithms for maximum flow in undirected graphs, and Madry's faster algorithm for maximum weight matching.

Bibliography

1. *Convex Optimization*, S. Boyd and L. Vandenberghe. Cambridge University Press. (pdf available online.)

2. *Introductory Lectures on Convex Optimization: A Basic Course.* Y. Nesterov. Springer 2004.

3. *Online Convex Programming and Generalized Infinitesimal Gradient Ascent.* M. Zinkevich, ICML 2003.

4. Book draft *Online convex optimization.* Elad Hazan.

5. Lecture notes on online optimization. S. Bubeck.

6. *Potential-Function Proofs for Gradient Methods.* Nikhil Bansal and Anupam Gupta.