CS5339 Lecture Notes #6b: Gradient-Based Optimization Algorithms

Jonathan Scarlett

March 31, 2021

Useful references:

- Blog post on gradient-based algorithms[1]
- Part I of Boyd and Vandenberghe's "Convex Optimization" book[2]
- Chapter 14 of "Understanding Machine Learning" book
- Duchi's machine learning summer school video on optimization[3]
- (Advanced) Lecture videos by Constantine Caramanis[4]

1 Introduction

- Optimization is central to the majority of machine learning algorithms, both classical and modern. Specifically, such algorithms typically seek to solve a problem of the following form (or similar):

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \, \frac{1}{n}\sum_{t=1}^{n} \text{Loss}(\mathbf{x}_t, y_t; \boldsymbol{\theta})$$

  for some vector of parameters $\boldsymbol{\theta}$ (possibly much more complex than just linear classification/regression parameters – e.g., neural network weights) and loss function $\text{Loss}(\cdot)$ (e.g., squared-loss in regression).

- In this lecture, we look at the question of *how to numerically solve such a minimization problem.*

- We will focus on minimization problems taking the above form, but the same ideas apply if we have

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \, \frac{1}{n}\sum_{t=1}^{n} \text{Loss}(\mathbf{x}_t, y_t; \boldsymbol{\theta}) + \lambda \cdot \text{Regularization}(\boldsymbol{\theta}),$$

  e.g., with squared ℓ_2-regularization of the form $\|\boldsymbol{\theta}\|^2$.

[1]https://ruder.io/optimizing-gradient-descent/
[2]http://web.stanford.edu/~boyd/cvxbook/
[3]https://sites.google.com/view/mlss-2019/lectures-and-tutorials
[4]https://www.youtube.com/playlist?list=PLXsmhnDvpjORzPelSDs0LSDrfJcqyLlZc

- To simplify the notation, we will switch to studying a generic optimization problem of the form

$$\text{minimize}_{\mathbf{x}}\ f(\mathbf{x}), \quad \text{where } f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}), \tag{1}$$

  with $f_i(\cdot)$ being generic functions, and $\mathbf{x} \in \mathbb{R}^d$ being the vector of parameters being optimized (not to be confused with $\mathbf{x}_t$ in classification and regression – rather, the $\mathbf{x}$ here plays the role of $\boldsymbol{\theta}$ there).

- The algorithms that we can discuss can also be applied to more general $f(\mathbf{x})$ that don't decompose into a sum from $i = 1$ to n, but this decomposition is so ubiquitous in machine learning that it deserves special attention.

2 Gradient Descent and its Variants

In this section, we assume that each f_i (and hence f) is differentiable. We will discuss non-differentiable functions in the next section.

Gradient descent.

- The idea of gradient descent is simple: Picturing the function being optimized as a "landscape", and starting in some initial location, try to repeatedly "step downhill" until the minimum is reached.

- Formally, the parameters $\mathbf{x}$ are repeatedly updated as follows:

$$\mathbf{x}_{\text{next}} = \mathbf{x} - \eta \cdot \nabla f(\mathbf{x}),$$

  where $\nabla f = \big[\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_d}\big]^T$ is the gradient vector associated with f, and $\eta > 0$ is a step size (i.e., dictating how large the steps are that we are taking).

  – In what sense is moving in the $-\nabla f$ moving downhill? To answer this, consider taking a very small step from $\mathbf{x}$ in some direction $\boldsymbol{\Delta}$ (with $\|\boldsymbol{\Delta}\| = 1$) to obtain $\mathbf{x} + \epsilon\boldsymbol{\Delta}$, where $\epsilon \ll 1$.

  – By a first-order Taylor expansion, we have

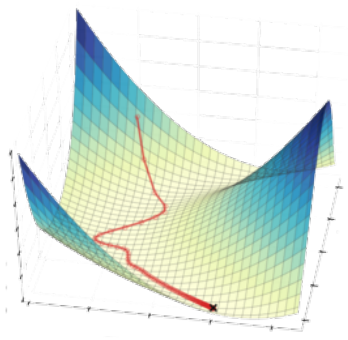
$$f(\mathbf{x} + \epsilon\boldsymbol{\Delta}) = f(\mathbf{x}) + \epsilon\langle \nabla f, \boldsymbol{\Delta}\rangle + O(\epsilon^2).$$

  – Since $\epsilon \ll 1$, we can treat the $O(\epsilon^2)$ term as negligible. So if we want $f(\mathbf{x} + \epsilon\boldsymbol{\Delta})$ to be as small as possible, then we want $\langle \nabla f(\mathbf{x}), \boldsymbol{\Delta}\rangle$ to be as negative as possible.

  – But the Cauchy-Schwartz inequality states that $|\langle \nabla f(\mathbf{x}), \boldsymbol{\Delta}\rangle| \le \|\nabla f(\mathbf{x})\| \cdot \|\boldsymbol{\Delta}\|$, with equality if and only if $\boldsymbol{\Delta} = c\nabla f(\mathbf{x})$ for some constant c.

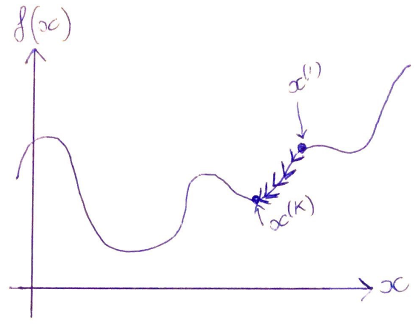
  – Hence, the most negative value of $\|\nabla f(\mathbf{x})\| \cdot \|\boldsymbol{\Delta}\| = -\|\nabla f(\mathbf{x})\|$ (recall that $\|\boldsymbol{\Delta}\| = 1$) is obtained when $\boldsymbol{\Delta}$ points in the $-\nabla f$ direction.

- An illustration of minimizing a 2D function ($d = 2$):



  - The algorithm starts near the top of the figure, then slowly "steps downhill" until it arrives at the black 'X' point.

- Of course, in general, this strategy could get stuck in a suboptimal *local minimum*:



On the other hand, if $f(\mathbf{x})$ is a convex function, then there is any local minimum is guaranteed to be a global minimum.

Stochastic gradient descent.

- Substituting $f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$ into the gradient descent update equation gives

$$\mathbf{x}_{\text{next}} = \mathbf{x} - \eta \cdot \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{x}),$$

  since the gradient of a sum is the sum of gradients.

- For large n (i.e., "big data", as is often the case in modern applications), this causes computational headaches – we sum n terms on every iteration, but each step might not actually be gaining us much!

- A more efficient (but "noisier") alternative for large n is *stochastic gradient descent*:

$$\mathbf{x}_{\text{next}} = \mathbf{x} - \eta \cdot \nabla f_i(\mathbf{x}),$$

  where we have some method for sequentially selecting t (e.g., cycle through $\{1, \dots, n\}$ in order, or just perform uniformly random selection).

- Compared to gradient descent, the average over $i = 1, \dots, n$ is replaced by a single index i. So "on average" (with respect to t) we are still performing the correct update.
- As we will illustrate below, the SGD iterates can "jump around" more throughout the updates. In fact, this is not an entirely bad thing – it can help in escaping local minima.

- In practice, it is most common to take an approach in between the extremes of GD and SGD, and average over a *mini-batch* of indices, $\mathcal{I} \subset \{1, \dots, n\}$ of some fixed size (e.g., 5, 10, or 20). This leads to *mini-batch stochastic gradient descent*:

$$\mathbf{x}_{\text{next}} = \mathbf{x} - \eta \cdot \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \nabla f_i(\mathbf{x}),$$

  where again, there is some flexibility in how to choose $\mathcal{I}$ (e.g., it could contain a fixed number of randomly-chosen indices).

  - Setting $|\mathcal{I}| = n$ gives gradient descent, and setting $|\mathcal{I}| = 1$ gives stochastic gradient descent.

Example: Logistic regression

- Recall that in logistic regression, we wish to solve the following:

$$\min_{\boldsymbol{\theta}, \theta_0} \sum_{t=1}^{n} \log\big(1 + \exp(-y_t(\boldsymbol{\theta}^T \mathbf{x}_t + \theta_0))\big).$$

  Note that mapping this to our generic notation $f(\mathbf{x})$ requires substituting $\mathbf{x} = [\boldsymbol{\theta}^T \ \theta_0]^T$.

- The relevant derivatives are evaluated as follows (details omitted):

$$\frac{\partial}{\partial \theta_0} \log\big(1 + \exp(-y_t(\boldsymbol{\theta}^T \mathbf{x}_t + \theta_0))\big) = -y_t(1 - P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0))$$
$$\frac{\partial}{\partial \boldsymbol{\theta}} \log\big(1 + \exp(-y_t(\boldsymbol{\theta}^T \mathbf{x}_t + \theta_0))\big) = -y_t \mathbf{x}_t(1 - P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0)),$$

  where $\frac{\partial}{\partial \boldsymbol{\theta}}$ denotes a *gradient vector* (a length-d vector whose i-th entry contains the $\frac{\partial}{\partial \theta_i}$ term).

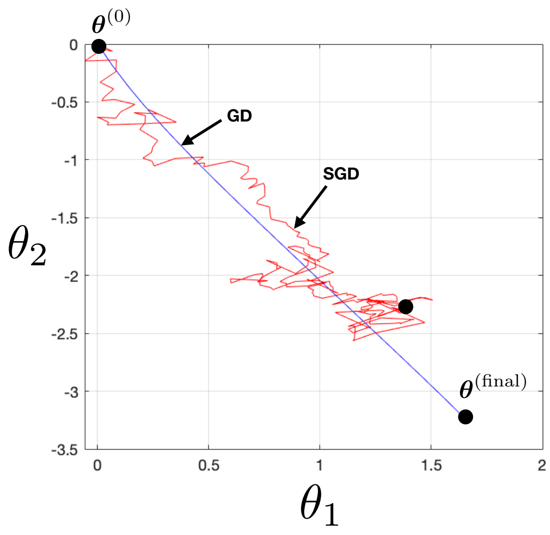
- Gradient descent updates:

$$\theta_{0,\text{next}} = \theta_0 + \eta \cdot \frac{1}{n} \sum_{t=1}^{n} y_t(1 - P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0))$$
$$\boldsymbol{\theta}_{\text{next}} = \boldsymbol{\theta} + \eta = \frac{1}{n} \sum_{t=1}^{n} y_t \mathbf{x}_t(1 - P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0))$$

- Stochastic gradient descent updates (without mini-batches):

$$\theta_{0,\text{next}} = \theta_0 + \eta \cdot y_t(1 - P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0))$$
$$\boldsymbol{\theta}_{\text{next}} = \boldsymbol{\theta} + \eta \cdot y_t \mathbf{x}_t(1 - P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0)).$$

- An example illustration of how the $\boldsymbol{\theta}$ parameters behave throughout the updates is as follows:

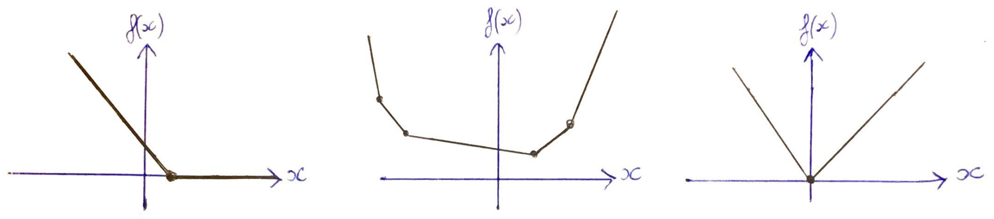


– Notice that both move solutions towards the same direction (namely, towards the $[1,-1]^T$ direction), but SGD does so in a much more erratic manner. We could make its behavior somewhat less erratic by using mini-batches and/or reducing the step size over time.

3 Non-Differentiable Functions

Non-differentiability and subgradients.

- Even in the special case of convex optimization, the objective function $f(\mathbf{x})$ may be non-differentiable. Some examples in the 1D case are as follows (e.g., hinge loss on the left):

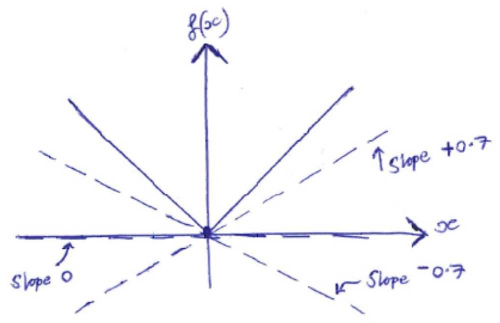


- Fortunately, for convex functions, we can introduce a generalized gradient-like notion that is well-defined at all points, and use it in place of the gradient.
- Specifically, for a convex function, the *subgradient* is defined as follows:

$$\partial f(\mathbf{x}) = \{\mathbf{g} \in \mathbb{R}^d \,:\, f(\mathbf{x}') \geq f(\mathbf{x}) + \langle \mathbf{g}, \mathbf{x}' - \mathbf{x} \rangle, \forall \mathbf{x}'\}$$

  – Note that $\partial f(\mathbf{x})$ is a set of vectors, not (necessarily) just a single vector
  – On the other hand, if f is differentiable at $\mathbf{x}$, then $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$, i.e., the gradient is the only element in the set defining the subgradient.

- Example: In a 1D setting, suppose that $f(x) = |x|$. Then for $x < 0$ we have $\partial f(x) = -1$, for $x > 0$ we have $\partial f(x) = 1$, and for $x = 0$ we have $\partial f(x) = [-1, 1]$. This is because for any $m \in [-1, 1]$, there exists a slope of gradient m passing through $(0,0)$ and lying below the entire function. An illustration:



Subgradient-based optimization.

- The *subgradient method* iteratively updates as follows:

$$\mathbf{x}_{\text{next}} = \mathbf{x} - \eta \mathbf{g},$$

  where $\mathbf{g}$ is any subgradient of $\mathbf{x}$ (i.e., $\mathbf{g} \in \partial f(\mathbf{x})$).

- If f is differentiable, then this reduces to regular gradient descent, since $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$. Stochastic variants also follow in the same way as above.

- More generally, the idea is clearly similar, in the sense of trying to step downhill. However, while gradient descent can be shown to literally descend (i.e., $f(\mathbf{x}_1) \ge f(\mathbf{x}_2) \ge \dots$) under mild technical conditions, this is not always the case for the subgradient method (see the tutorial).

- Hence, instead of just returning the final $\mathbf{x}$ (denoted by $\mathbf{x}_K$ after K iterations), it may be better to keep track of all $\mathbf{x}_1, \dots, \mathbf{x}_K$ and return the one with the lowest $f(\cdot)$ value.

4 Convergence Analysis

Notes.

- In general, proving that an optimization algorithm converges (ideally to the optimal value) requires making *smoothness* assumptions on the function f. Common assumptions include differentiability, convexity, Lipschitz continuity, strong convexity, and less straightforward notions such as self-concordance (we will not define these here).

- Here we present just one example of a convergence analysis, considering a slight variation of the subgradient method in which the step size may vary from iteration to iteration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \mathbf{g}_k,$$

  where $\mathbf{g}_k$ is any subgradient of $\mathbf{x}_k$.

Formal statement.

- We make the following assumptions:

  1. f is convex;
  2. There exists some $\mathbf{x}^*$ achieving $\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$;
  3. Lipschitz condition: $\|\mathbf{g}\| \le M$ for any subgradient $\mathbf{g}$ corresponding to any $\mathbf{x}$;
  4. The initialization $\mathbf{x}^{(1)}$ satisfies $\|\mathbf{x}^{(1)} - \mathbf{x}^*\| \le R$ for some finite R.

- **Theorem.** Under the preceding assumptions, using the subgradient method with any sequence of step sizes $\{\eta_k\}_{k=1}^\infty$ satisfying $\lim_{k\to\infty}\eta_k = 0$ and $\lim_{K\to\infty}\sum_{k=1}^\infty \eta_k = \infty$, we have

$$\min_{k=1,\dots,K} f(\mathbf{x}_K) \to f(\mathbf{x}^*)$$

  as $k \to \infty$.

- In optimization theory, one is not only interested in converging to $f(\mathbf{x}^*)$, but also the *convergence rate* (i.e., how fast it gets there). This is discussed briefly after the proof.

Proof of the theorem:

- We start by establishing how close the $(k+1)$-th iterate is to $\mathbf{x}^*$ as a function of the k-th iterate:

$$\begin{aligned}
\frac{1}{2}\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 &\overset{(a)}{=} \frac{1}{2}\|\mathbf{x}_k - \eta_k\mathbf{g}_k - \mathbf{x}^*\|^2 \\
&\overset{(b)}{=} \frac{1}{2}\|\mathbf{x}_k - \mathbf{x}^*\|^2 - \eta_k\mathbf{g}_k^T(\mathbf{x}_k - \mathbf{x}^*) + \frac{\eta_k^2}{2}\|\mathbf{g}_k\|^2 \\
&\overset{(c)}{\le} \frac{1}{2}\|\mathbf{x}_k - \mathbf{x}^*\|^2 - \eta_k(f(\mathbf{x}_k) - f(\mathbf{x}^*)) + \frac{\eta_k^2}{2}\|\mathbf{g}_k\|^2.
\end{aligned}$$

  where (a) substitutes the subgradient method's update rule, (b) expands the square, and (c) uses the definition of subgradient for convex functions, i.e., $f(\mathbf{x}_k) \ge f(\mathbf{x}^*) + \mathbf{g}_k^T(\mathbf{x}_k - \mathbf{x}^*)$ (to visualize in the 1D case, this is just the fact that the tangent curve lies below the function).

- Re-arranging the above gives

$$\eta_k(f(\mathbf{x}_k) - f(\mathbf{x}^*)) \le \frac{1}{2}\|\mathbf{x}_k - \mathbf{x}^*\|^2 - \frac{1}{2}\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 + \frac{\eta_k^2}{2}\|\mathbf{g}_k\|^2,$$

  and summing from $k = 1$ to any fixed iteration index K gives

$$\begin{aligned}
\sum_{k=1}^K \eta_k(f(\mathbf{x}_k) - f(\mathbf{x}^*)) &\overset{(d)}{\le} \frac{1}{2}\|\mathbf{x}_1 - \mathbf{x}^*\|^2 - \frac{1}{2}\|\mathbf{x}_{K+1} - \mathbf{x}^*\|^2 + \sum_{k=1}^K \frac{\eta_k^2}{2}\|\mathbf{g}_k\|^2 \\
&\overset{(e)}{\le} \frac{1}{2}\|\mathbf{x}_1 - \mathbf{x}^*\|^2 + \sum_{k=1}^K \frac{\eta_k^2}{2}\|\mathbf{g}_k\|^2, \\
&\overset{(e)}{\le} \frac{1}{2}R^2 + \frac{1}{2}M^2\sum_{k=1}^K \eta_k^2,
\end{aligned}$$

  where (d) uses the fact that the addition (first term) and subtraction (second term) of each $\|\mathbf{x}_i - \mathbf{x}^*\|^2$ cancels to zero (except for $i = 1$ and $i = K+1$), (e) applies the trivial bound $\|\cdot\|^2 \ge 0$, and (f) uses the assumptions $\|\mathbf{g}\| \le M$ and $\|\mathbf{x}_1 - \mathbf{x}^*\| \le R$.

- Now consider what the *best* $\mathbf{x}_k$ *found so far* is. Since the minimum is certainly less than (or at least no higher than) any weighted average, we have

$$\Big(\sum_{k=1}^K \eta_k\Big)\Big(\min_{k=1,\dots,K}(f(\mathbf{x}_k)-f(\mathbf{x}^*))\Big) \le \sum_{k=1}^K \eta_k(f(\mathbf{x}_k)-f(\mathbf{x}^*)),$$

and combining this with the above inequality gives

$$\min_{k=1,\dots,K}(f(\mathbf{x}_k)-f(\mathbf{x}^*)) \le \frac{\frac{1}{2}R^2+\frac{1}{2}M^2\sum_{k=1}^K\eta_k^2}{\sum_{k=1}^K\eta_k}.$$

- It is straightforward to show that if the sequence η_k satisfies both $\eta_k \to 0$ (as $k\to\infty$) and $\sum_{k=1}^K \eta_k \to \infty$ (as $K\to\infty$), then the ratios $\frac{\frac{1}{2}R^2}{\sum_{k=1}^K\eta_k}$ and $\frac{\frac{1}{2}M^2\sum_{k=1}^K\eta_k^2}{\sum_{k=1}^K\eta_k}$ both approach zero as $K\to\infty$ (the former is immediate, and the letter is essentially because $\eta^2 \ll \eta$ when η is small). Hence,

$$\min_{k=1,\dots,K}(f(\mathbf{x}_k)-f(\mathbf{x}^*)) \to 0 \quad \text{as } K\to\infty$$

which proves the theorem.

Convergence rate and extensions.

- Some choices of $\{\eta_k\}$ are investigated in the tutorial, with a good choice being $\eta_k = \frac{\eta_0}{\sqrt{k}}$, and yielding a convergence to zero at rate $O\big(\frac{\log K}{\sqrt{K}}\big)$.
- A sharper analysis can also give $O\big(\frac{1}{\sqrt{K}}\big)$, which is in fact the best possible for the standard subgradient method without further assumptions. On the other hand, if further smoothness assumptions are imposed (e.g., differentiability, strong convexity, etc.) and/or a more sophisticated gradient algorithm is used, then improvements such as $O\big(\frac{1}{K}\big)$ or even $O(e^{-cK})$ are possible.
- The above analysis can be extended to stochastic versions of the subgradient method (i.e., $\mathbf{g}_k$ is random and satisfies $\mathbb{E}[\mathbf{g}_k|\mathbf{x}_k] \in \partial f(\mathbf{x}_k)$), and a projected version for constrained optimization (see below).

5 More Advanced Variants

Projected gradient methods for constrained problems:

- As we discussed in the previous lecture, optimization problems are often *constrained*, taking the form

$$\text{minimize } f(\mathbf{x}) \quad \text{subject to } \mathbf{x}\in\mathcal{C},$$

where $\mathcal{C}$ is some constraint set. For instance, in the formulation of the previous lecture, the set would be $\mathcal{C} = \{\mathbf{x}\in\mathbb{R}^d \,:\, f_i(\mathbf{x}) \le 0, \forall i \text{ and } h_i(\mathbf{x}) = 0, \forall i\}$.Li

- In this case, we can easily modify the above gradient-based methods to the following:

$$\mathbf{x}_{\text{next}} = \Pi_{\mathcal{C}}(\mathbf{x}-\eta\mathbf{g}),$$

where $\mathbf{g} = \nabla f(\mathbf{x})$ (gradient descent) or $\mathbf{g}\in\partial f(\mathbf{x})$ (subgradient descent), and $\Pi_{\mathcal{C}}(\cdot)$ is a projection operator onto the set $\mathcal{C}$. That is, $\Pi_{\mathcal{C}}(\mathbf{x}')$ is the point in $\mathcal{C}$ that is closest to $\mathbf{x}'$.

  - Example: If $\mathcal{C}$ is the set of $\mathbf{x}$ satisfying $a_i \le x_i \le b_i$ for $i = 1, \dots, d$, then the projection operation keeps the i-th coordinate unchanged if it is already in $[a_i, b_i]$, rounds up to a_i when below this range, or rounds down to b_i when above this range.

- The convergence analysis of the previous section easily extends to this variant with projection, under the same assumptions along with the assumption that $\mathcal{C}$ is a convex set. In fact, we can fairly easily additionally generalize from subgradients ($\mathbf{g} \in \partial f(\mathbf{x})$) to stochastic subgradients ($\mathbf{g}$ is random and $\mathbb{E}[\mathbf{g}|\mathbf{x}] \in \partial f(\mathbf{x})$) – see the tutorial.

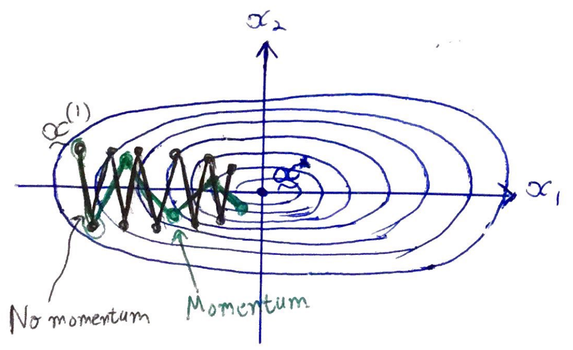
Other more advanced variants. We have only scratched the surface of optimization algorithms, and the topic could easily fill multiple entire courses. Below, just a small sample of additional concepts are outlined that we do not cover:

- Line search: Instead of specifying a fixed step size or a fixed sequence, one can use a method called *line search* to try to automatically take a good step size on each iteration (essentially, trying multiple step sizes and choosing a good one based on the resulting function values). This can be more effective, but also increases the computation.

- Momentum: One way to overcome erratic/oscillatory behavior is through the use of *momentum*, which updates as follows for some $\gamma \in [0, 1]$:

$$\begin{aligned} \mathbf{v}_k &= \eta \nabla f(\mathbf{x}_k) + \gamma \mathbf{v}_{k-1} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k - \eta \mathbf{v}_k. \end{aligned}$$

  - Setting $\gamma = 0$ recovers regular gradient descent, whereas typically γ is around 0.9.
  - The inclusion of $\gamma \mathbf{v}_{k-1}$ amounts to remembering part of the previous update and including it.
  - Intuitively, if the solution is jumping back and forth among some direct, then combining the new gradient $\nabla f(\mathbf{x}_k)$ with the previous term $\gamma \mathbf{v}_{k-1}$ helps to "dampen" this (e.g., negative and positive terms cancel). An illustration (with ellipses representing "level sets", like on a contour map)



- Further algorithms: More sophisticated gradient-based methods are frequently used in the optimization of neural networks (e.g., Adagrad, RMSprop, Adam). See `https://ruder.io/optimizing-gradient-descent/` for a useful summary.

- Distributed algorithms: A variety of parallel and distributed optimization algorithms exist, which are beneficial when the required computation is to expensive for a single machine.

- Second-order methods: Second-order methods exist that exploit second-derivative information (e.g., Newton or Newton-like methods). Standard forms of these are not suited to large-scale machine learning, since a d-dimensional function has $O(d^2)$ second derivatives (as opposed to only $O(d)$ gradients) of the form $\frac{\partial^2 f}{\partial x_i \partial x_j}$, but lower-computation approximations can still be suitable.

- Zero-th order methods: In some cases, we don't have the luxury of having access to gradients (e.g., because the function value itself is the output of a procedure that has no mathematical description to "take the derivative of"). For such settings, there exist *zero-th order methods* (also known as *black-box optimization*) that optimize using only queries to $f(\cdot)$.