Chapter 5

So far, we have focused on methods that are based on the notion of gradient and gradient descent: at every iteration, we compute first-order (=gradient) information about the objective, and we use this information to perform an educated step towards a local or a global minimum of the objective. We have shown—through theoretical analysis—what we can achieve by using gradients, and what is the best we can hope for (=lower bounds).
But, what are some ways to accelerate, both in terms of iteration and analytical complexity, this first set of algorithms? We will present some approaches that deviate from simple gradient-based methods, and that provably or empirically outperform the so-far studied methods: these include Newton's method, and quasi-Newton variants. To complete the picture on the theory side, we will continue in the convex world and compare the asymptotic bounds to understand what we gain and what we lose for each of these choices.

Newton's method | quasi-Newton variants | Natural gradient

We remind first what are the limits of gradient descent. The following summarize lower bounds we can expect, *by only using gradients in optimizations*, from some of the types of objective functions we have previously discussed.

- For objective functions with Lipschitz continuous gradients, with constant L, we can prove that:
$$f(x_T) - f(x^\star) \geq \frac{3L\|x_0 - x^\star\|_2^2}{32(T+1)^2} = O\left(\tfrac{1}{T^2}\right).$$
Under this assumption, and only using gradients, we cannot achieve better than the above.
- For objectives functions with both Lipschitz continuous gradients and strong convexity:
$$\|x_T - x^\star\|_2^2 \geq \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2T} \|x_0 - x^*\|_2^2.$$
where $\kappa = L/\mu > 1$. Here we observe that, while we have achieved the same convergence rate with respect to the exponent—i.e., in both cases we have c^T, for $c < 1$—in the lower bound case, we see $\sqrt{\kappa}$ instead of κ.

But how we obtain such lower bounds? Can we achieve something better? This chapter follows a path on different algorithmic approaches, some of which deviate from just using first-order methods, in order to show what we gain (and lose) in practice.

The (notorious) Newton's method. Newton's method has been and still is one of the most celebrated algorithms in numerical scientific computing. The reason we target numerical scientific computing as an example here is because there are applications where we care getting a solution to a problem accurately: e.g., there might be cases where the accuracy of estimation at the error level of 10^{-15} is what is important for the algorithm; so, getting a solution that is just 10^{-4} close might be unacceptable.

Let us first derive the Newton's iteration. Newton's method, as a descent method, updates the current estimate x as:
$$x \leftarrow x + \Delta x,$$
where Δx abstractly defines a direction/update that moves x to a "better place", with respect to the objective we try to minimize. In this chapter, we focus on the unconstrained case:
$$\min_{x \in \mathbb{R}^p} \quad f(x),$$
where f is assumed to be twice differentiable, with gradient $\nabla f(\cdot)$ and Hessian $\nabla^2 f(\cdot)$. By taking the second-order Taylor expansion of f around x, we have:
$$\begin{aligned} f(x+\Delta x) &\approx f(x) + \langle \nabla f(x), (x+\Delta x) - x\rangle \\ &\quad + \tfrac{1}{2}\left\langle \nabla^2 f(x)\left((x+\Delta x) - x\right), \left((x+\Delta x)-x\right)\right\rangle \\ &\approx f(x) + \langle \nabla f(x), \Delta x\rangle + \tfrac{1}{2}\left\langle \nabla^2 f(x)\Delta x, \Delta x\right\rangle \end{aligned}$$
(*Similar reasoning is used for gradient descent to connect Δx with negative gradient, $-\nabla f(\cdot)$, as the best descent direction for first-order methods.*) Using this characterization, we can locally find the best Δx by finding the root of the quadratic approximation; i.e., by finding Δx that makes the gradient of $f(x+\Delta x)$ be zero:
$$\nabla_{\Delta x} f(x+\Delta x) = 0 \quad \overset{\text{approx.}}{\Rightarrow} \quad \nabla f(x) + \nabla^2 f(x)\Delta x = 0$$
$$\Rightarrow \quad \Delta x = -\left(\nabla^2 f(x)\right)^{-1}\nabla f(x).$$
Substituting Δx in $x + \Delta x$, we obtain the Newton's iteration:
$$x_{t+1} = x_t - \eta_t H_t^{-1}\nabla f(x_t), \quad H_t := \nabla^2 f(x_t).$$
(*As we will show, $\eta_t = 1$ in theory; however, there are cases that require $\eta_t < 1$ at least at the beginning of the algorithm, or when we initialize badly the algorithm. When $\eta_t < 1$, we call the method* damped Newton's method, *and its study is outside the scope of this course.*)

Before we present some theory for Newton's method, we need to get the full picture of what we are proposing: with Newton's method, we actually do gradient descent, but before applying the gradient $\nabla f(x_t)$, we "translate" it through the matrix $\nabla^2 f(x_t)^{-1}$; i.e., we use the transformed gradient $\widetilde{\nabla} f(x_t) := \nabla^2 f(x_t)^{-1}\nabla f(x_t)$.

Let us first study the behavior of Newton's method in general, even non-convex, scenarios.

Theorem 2. *Let $\min_x f(x)$ be the problem of interest, with f being twice differentiable, f has Lipschitz continuous Hessians:*
$$\left\|\nabla^2 f(x) - \nabla^2 f(y)\right\|_2 \leq M \cdot \|x-y\|_2,$$
and f satisfies at the optimum point $x^\star$: $\nabla^2 f(x^\star) \succeq \mu I$. Assuming that we start from a point x_0 that is close enough to:
$$\|x_0 - x^\star\|_2 < \tfrac{2\mu}{3M},$$
Newton's method:
$$x_{t+1} = x_t - \eta_t H_t^{-1}\nabla f(x_t), \quad H_t := \nabla^2 f(x_t),$$
converges according to:
$$\|x_{t+1} - x^\star\|_2 \leq \frac{M\|x_t - x^\star\|_2^2}{2\left(\mu - M\|x_t - x^\star\|_2\right)}.$$

Proof: By the main recursion of Newton's method, we have:
$$\begin{aligned} x_{t+1} - x^\star &= x_t - \left(\nabla^2 f(x_t)\right)^{-1}\nabla f(x_t) - x^\star \\ &= x_t - \left(\nabla^2 f(x_t)\right)^{-1} \cdot \left(\int_0^1 \nabla^2 f\left(x^\star + \tau(x_t - x^\star)\right)\cdot(x_t - x^\star)d\tau\right) - x^\star \\ &= (x_t - x^\star) - \left(\nabla^2 f(x_t)\right)^{-1} \cdot \left(\int_0^1 \nabla^2 f\left(x^\star + \tau(x_t - x^\star)\right)\cdot(x_t - x^\star)d\tau\right) \\ &= \left(\nabla^2 f(x_t)\right)^{-1} \cdot G_t \cdot (x_t - x^\star) \end{aligned}$$
where
$$G_t = \int_0^1 \left(\nabla^2 f(x_t) - \nabla^2 f\left(x^\star + \tau(x_t - x^\star)\right)\right)d\tau.$$

The integral above comes from the application the Taylor's theorem / mean value theorem.

We proceed by bounding the terms on the right hand side. (*Remember that $\|\cdot\|_2$ with matrices corresponds to the spectral norm, not the Frobenius norm.*)

$$\begin{aligned}\|G_t\|_2 &= \left\| \int_0^1 \left(\nabla^2 f(x_t) - \nabla^2 f\left(x^\star + \tau(x_t - x^\star)\right)\right) d\tau \right\|_2 \\ &\leq \int_0^1 \left\|\nabla^2 f(x_t) - \nabla^2 f\left(x^\star + \tau(x_t - x^\star)\right)\right\|_2 d\tau \\ &\leq \int_0^1 M \cdot \|x_t - x^\star + \tau(x_t - x^\star)\|_2 d\tau \\ &= \tfrac{M\|x_t - x^\star\|_2}{2}\end{aligned}$$

Moreover, we know that, by the Hessian Lipschtiz continuity:

$$\left\|\nabla^2 f(x) - \nabla^2 f(y)\right\|_2 \leq M \cdot \|x - y\|_2,$$

we have:

$$\nabla^2 f(x) - M\|x-y\|_2 \cdot I \preceq \nabla^2 f(y) \preceq \nabla^2 f(x) + M\|x-y\|_2 \cdot I,$$

$\forall x, y$, and thus holds for $x = x_t$ and $y = x^\star$:

$$\nabla^2 f(x_t) \succeq \nabla^2 f(x^\star) - M\|x_t - x^\star\|_2 \cdot I \succeq (\mu - M\|x_t - x^\star\|_2) \cdot I.$$

Assume that $\|x_t - x^\star\|_2 \leq \frac{\mu}{M}$ (*to be justified a posteriori*), we have:

$$\left\|\nabla^2 f(x_t)^{-1}\right\|_2 \leq (\mu - M\|x_t - x^\star\|_2)^{-1}.$$

Combining all the above, we get:

$$\begin{aligned}\|x_{t+1} - x^\star\|_2 &\leq \left\|\left(\nabla^2 f(x_t)\right)^{-1} \cdot G_t \cdot (x_t - x^\star)\right\|_2 \\ &\leq \left\|\nabla^2 f(x_t)^{-1}\right\|_2 \cdot \|G_t\|_2 \cdot \|x_t - x^\star\|_2 \\ &\leq (\mu - M\|x_t - x^\star\|_2)^{-1} \cdot \tfrac{M\|x_t - x^\star\|_2}{2} \cdot \|x_t - x^\star\|_2 \\ &= \frac{M\|x_t - x^\star\|_2^2}{2\left(\mu - M\|x_t - x^\star\|_2\right)}.\end{aligned}$$

Let us discuss the initialization assumption: $\|x_0 - x^\star\|_2 \leq \frac{2\mu}{3M}$. Using induction, we have the following two steps.

`Basis step:` We have:

$$\begin{aligned}\|x_1 - x^\star\|_2 &\leq \frac{M\|x_0 - x^\star\|_2^2}{2\left(\mu - M\|x_0 - x^\star\|_2\right)} \\ &= \frac{M \cdot \frac{4\mu^2}{9M^2}}{2\left(\mu - M \cdot \frac{2\mu}{3M}\right)} \\ &= \frac{\frac{4\mu^2}{9M}}{2\left(\frac{3M\mu - 2M\mu}{3M}\right)} = \frac{\frac{4\mu^2}{9M}}{\frac{2\mu}{3}} = \frac{2\mu}{3M}.\end{aligned}$$

`Induction step:` Assume that for some t, it holds $\|x_t - x^\star\|_2 \leq \frac{2\mu}{3M}$. This also justifies the assumption that $\|x_t - x^\star\|_2 \leq \frac{2\mu}{3M} \leq \frac{\mu}{M}$ which is used in the proof above as an assumption. Then:

$$\begin{aligned}\|x_{t+1} - x^\star\|_2 &\leq \frac{M\|x_t - x^\star\|_2^2}{2\left(\mu - M\|x_t - x^\star\|_2\right)} \\ &= \dots = \tfrac{2\mu}{3M}.\end{aligned}$$

This completes the proof: i.e., assuming a good enough initialization, $\|x_0 - x^\star\|_2 \leq \frac{2\mu}{3M}$, all the assumptions in the proof are justified, leading the recursion in the theorem. □

Before we proceed, let's first understand what this recursion mean. By assumption of initialization, the recursion becomes:

$$\begin{aligned}\|x_{t+1} - x^\star\|_2 &\leq \frac{M\|x_t - x^\star\|_2^2}{2\left(\mu - M\|x_t - x^\star\|_2\right)} \\ &\leq \frac{M\|x_t - x^\star\|_2^2}{2\left(\mu - M\frac{2\mu}{3M}\right)} \\ &= \tfrac{3M}{2\mu} \cdot \|x_t - x^\star\|_2^2 \equiv c \cdot \|x_t - x^\star\|_2^2.\end{aligned}$$

Under the assumption that we start from a good initialization point where $\|x_t - x^\star\|_2 \leq 1$—i.e., $\frac{2\mu}{3M} \leq 1$—this translates that the new distance is *quadratically* decreased, rather than linearly. That is, if we want $\|x_T - x^\star\|_2 \leq \varepsilon$, then this can be achieved in $O\left(\log\log\frac{1}{\varepsilon}\right)$ iterations. See also the convergence rate figure.

What if we assume convexity of f? It turns out that, using convexity, we do not gain anything in terms of convergence rate. However, assuming convexity, we can achieve *global* convergence: irrespective of the initialization, there is analysis that proves that Newton's method converges to the global minimum. *There is a caveat though:* The quadratic convergence rate holds only locally! I.e., we are guaranteed quadratic convergence rate, after we perform some steps with slower rate. Only after we get inside a region, close enough to the global minimum, the quadratic rate is activated!

Some comments on Newton's method.

- Newton's method exploits local curvature of the function. This is depicted in the following figures, borrowed from Boyd's and Vandenberghe's book. In the first case, gradient descent method is myopic and gradient suggests a direction that is almost perpendicular to the direction we should move.

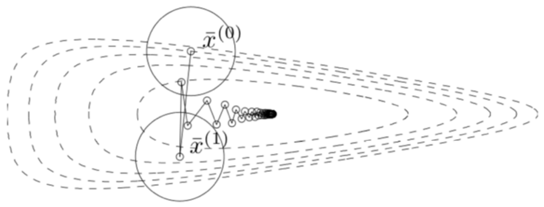

Fig. 28. Gradient descent behavior in function valleys.

On the other hand, Newton's method "warps" the function landscape, where gradient direction moves more towards the optimum.

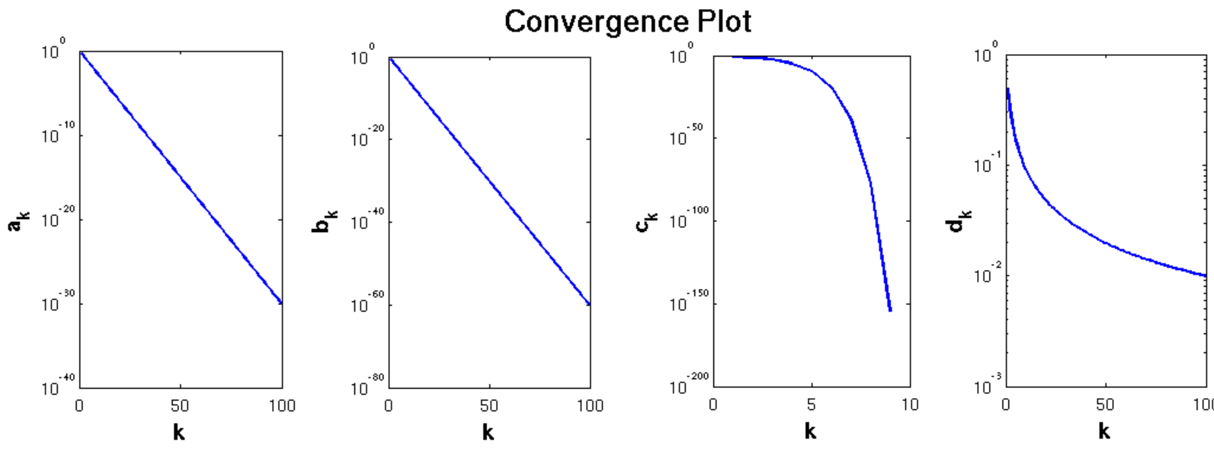

Fig. 30. Borrowed from Wikipedia. Illustration of different convergence rates. Note that y-axis is in logarithmic scale for all the plots, while the x-axis has a linear scale. The y-axis denotes a metric that dictates to the optimum point; think for example $\|x_k - x^\star\|_2$ (*We use k as an iteration subscript here*). The x-axis represent the iteration count k. The first two plots represent *linear* convergence rates: it is called linear as a convention to match the linear curve in the *logarithmic* y-axis scale. While the second plot depicts a more preferable behavior, in the big-Oh notation, the two plots are equivalent. For an error level ε, linear convergence rate implies $O\left(\log \frac{1}{\varepsilon}\right)$. The third plot depicts a *quadratic* convergence rate. For an error level ε, linear convergence rate implies $O\left(\log\log \frac{1}{\varepsilon}\right)$. Finally, the fourth plot represents the *sublinear* convergence rate; much slower than the linear rate. Some typical rates are: $O\left(1/\varepsilon^2\right)$, $O\left(1/\varepsilon\right)$, $O\left(1/\sqrt{\varepsilon}\right)$.

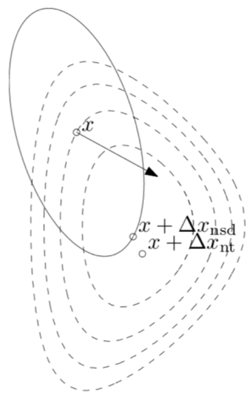

Fig. 29. Newton's method first changes the function landscape, and then performs gradient descent on this space.

- Each iteration of Newton's method is *more expensive computationally* than simple gradient descent. Thus, there is a trade-off: while we need much less number of iterations to get to optimum (after good initialization), we pay much more per iteration. (*Think of the case where computing the Hessian does not fit in computer's main memory*). Remember that if $\nabla f(x) \in \mathbb{R}^p$, then $\nabla^2 f(x) \in \mathbb{R}^{p\times p}$; if $\nabla f(X) \in \mathbb{R}^{p\times p}$, then $\nabla^2 f(X) \in \mathbb{R}^{p^2\times p^2}$; putting $p = 10^6$, we get an idea of how things could scale in practice.
- Theory so far assumes a good initialization point to achieve quadratic convergence rate.
- Newton's method is rarely used in machine learning applications because we often do not care about exact solutions; Newton's method is extremely important in cases where accuracy is key, such as numerical analysis and scientific computing.
- Comparing to what we can achieve with gradient descent, Newton's method "breaks" the lower bound

$$\|x_T - x^\star\|_2^2 \geq \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2T} \|x_0 - x^*\|_2^2 \equiv c^T \cdot \|x_0 - x^*\|_2^2,$$

since we actually have:

$$\|x_T - x^\star\|_2^2 \leq c^T \cdot \|x_0 - x^*\|_2^4.$$

Spanning the space between gradient descent and Newton's method. Newton's method proposes a different way of performing gradient descent: instead of just taking the gradient per iteration, we compute the Hessian to weigh the gradient. This raises the question: *Does only the true Hessian work as a weighting factor for the gradient? Can we generate another H_t in:*

$$x_{t+1} = x_t - \eta_t H_t^{-1} \nabla f(x_t).$$

This leads to the class of general preconditioning matrices and preconditioning methods. Often these methods are also called quasi-Newton methods, as we do not use the exact Hessian information per iteration.

We will use the following general rule for quasi-Newton methods: For $x_t \in \mathbb{R}^p$, we have

$$x_{t+1} = x_t - \eta_t H_t^{-1} \nabla f(x_t), \quad H_t \in \mathbb{R}^{p\times p}.$$

There are numerous ways to perform this step—i.e., there are various ways to generate H_t per iteration—but we will focus on two of them for now (*More to be discussed as we proceed.*):

- The (L)BFGS approximation;
- The SR1 approximation.

Both of them handle the unconstrained case:

$$\min_{x\in\mathbb{R}^p} f(x)$$

The Broyden-Fletcher-Goldfarb-Shanno approximation, a.k.a. BFGS. The BFGS approximation is based on the following reasoning:

- We know by Taylor's theorem that we can approximate the objective $f(\cdot)$ around x_t as:

  $$g_t(\Delta x) := f(x_t) + \langle \nabla f(x_t),\, \Delta x\rangle + \tfrac{1}{2}\langle H_t \Delta x,\, \Delta x\rangle,$$

  where H_t represents the actual Hessian matrix. Note here that $g_t(\Delta x)$ represents a local quadratic approximation of

f, and Δx is a vector that defines the direction we want to take: $x_{t+1} = x_t + \Delta x$. Thus, in an iterative fashion, we will generate the sequence $\dots, g_{t-1}(\cdot), g_t(\cdot), g_{t+1}(\cdot), \dots$, where at each iteration we compute a new Δx.

- Instead of using the exact Hessian in $H_t := \nabla^2 f(x_t)$, we look for an approximation of the Hessian. Remember that we use $g(\cdot)$ to compute the new Δx. We need some conditions that this function should satisfy:
  1. *When we take the gradient of $g_{t+1}(\cdot)$ at the zero point—meaning that we do not move at all—we should get back the gradient of the original function.* This condition makes sure that the quadratic approximation of f around its original point x_{t+1} gives back the original gradient of the function, $\nabla f(x_{t+1})$:
  $$\nabla g_{t+1}(0) = \nabla f(x_{t+1})$$
  2. *When we take the gradient of the new function approximation $g_{t+1}(\cdot)$, evaluated at the point after reversing the direction $-\Delta x$, then we should obtain back the gradient of f at the previous iteration.* I.e.,
  $$\nabla g_{t+1}(-\Delta x) = \nabla f(x_t)$$
  3. Inspired by making the local approximation quadratic, we also require per iteration to have:
  $$H_{t+1} \succ 0$$
- Let us use the above information to generate some useful equations. First, observe that by taking gradient of $\nabla g_{t+1}(-\Delta x)$ and using the above equation, we get:
$$\nabla g_{t+1}(-\Delta x) = \nabla f(x_t) \;\Rightarrow\; H_{t+1}\Delta x = \nabla f(x_{t+1}) - \nabla f(x_t)$$
This is known as the *secant equation*. Further, by the assumption that $H_{t+1} \succ 0$, the above becomes:
$$\langle \Delta x,\; \nabla f(x_{t+1}) - \nabla f(x_t)\rangle > 0.$$
Why? What is the intuition of this expression?
- The above lead to a recipe: per iteration, we are looking for a matrix $H_{t+1} \succ 0$ such that the secant equation is satisfied. But, *how many such H_{t+1} exist? Actually, infinite!* To restrict the search space, BFGS method solves the following optimization problem per iteration:
$$\begin{aligned} \min_{H \succ 0} \quad & \|H - H_t\|_F^2 \\ \text{subject to} \quad & H = H^\top \\ & H\Delta x = \nabla f(x_{t+1}) - \nabla f(x_t) \end{aligned}$$
Solving this problem (*we intentionally not solve it for now*), we obtain H_{t+1}; in order to use H_{t+1}, we further need to invert it and use it as:
$$x_{t+2} = x_{t+1} - \eta_{t+1} H_{t+1}^{-1} \nabla f(x_{t+1}).$$
I.e., we have found a way to compute a matrix H_{t+1}, but we still need to invert it, just like Newton's method! If this is the case, why don't we then just compute $\nabla^2 f(x_{t+1})$, which we know it is optimal?
- BFGS method goes a bit further to handle this case: Instead of compute in the H domain and then invert, we define $B := H^{-1}$, and we substitute that in the above expression.
$$\begin{aligned} \min_{B \succ 0} \quad & \|B - B_t\|_F^2 \\ \text{subject to} \quad & B = B^\top \\ & \Delta x = B\left(\nabla f(x_{t+1}) - \nabla f(x_t)\right) \end{aligned}$$
I.e., *we approximate the inverse directly so that $x_{t+1} = x_t - \eta_t B_t \nabla f(x_t)$!*
- But, how easy it is to solve the above problem? It is no luck (*remember that there are infinite matrices that satisfy what we need*) that the above problem has a closed form solution:
$$B_{t+1} = \left(I - \tfrac{s_t y_t^\top}{s_t^\top y_t}\right) B_t \left(I - \tfrac{y_t s_t^\top}{s_t^\top y_t}\right) + \tfrac{s_t s_t^\top}{s_t^\top y_t}$$
where
$$\begin{aligned} s_t &:= \Delta x \\ y_t &:= \nabla f(x_{t+1}) - \nabla f(x_t) \end{aligned}$$
- Some notes on the above: *i)* How do we initialize? I.e., how do we set B_0? Standard configurations assume $B_0 = I$. *ii)* What is the computational complexity of the above operations? First, observe that we have all the ingredients computed, as if we were performing gradient descet: at the $t+1$ iteration, we have $\nabla f(x_t)$ and $\nabla f(x_{t+1})$. Second, we perform only inner and outer product operations, which is often much faster than computing the actual Hessian and inverting it. Overall, BFGS is a method that uses only first-order information to approximate the second-order information.

The symmetric, rank-1 approximation, a.k.a. SR1. The BFGS method described above made no assumption about the function f, other than being differentiable. Thus, BFGS can be used both for convex and non-convex optimization, where at each iteration we force the secant equation + positive definiteness to find the new preconditioner. This means though that per iteration we approximate the function f with a second-order function that always looks upwards! I.e., per iteration we locally approximate f with a "bowl", even if f originally might look locally as a saddle. While this never happens in the convex case (and thus BFGS sounds like a great choice when we minimize a convex function), there might be cases where we minimize a non-convex function, and it would be great to have different approaches to handle these cases.

This is where SR1 approximation could be handy. As its name indicates, the SR1 approximation approximates a preconditioner matrix through successive rank-1 updates (*What kind of updates does BFGS?*). In particular, if H_t is the current approximation of second-order information, SR1 is based on the following approximation:
$$H_{t+1} = H_t + \sigma v v^\top,$$
for some vector v with appropriate dimensions, and $\sigma \in \{\pm 1\}$. *Key property is that such updates do not guarantee that the new approximation is positive definite, which could be a nice feature when we approximate non-convex functions.*

Assuming the secant equation is satisfied, the combination of the two equations leads to the following update:
$$B_{t+1} = B_t + \frac{(s_t - B_t y_t)(s_t - B_t y_t)^\top}{(s_t - B_t y_t)^\top y_t}.$$

Natural gradient: entering methodology in modern ML. Here, we will discuss the notion of *natural gradient*, relate it with the notion of Hessian in optimization, and set the scene for adaptive methods in training neural networks. To do so, we will need the following notions.

Let $\theta \in \mathbb{R}^p$ denote a set of variables that are unknown to us and we want to estimate. Here, we will follow more of a probabilistic approach where, given these parameters θ, we observe $x \in \mathbb{R}^d$, according to the distribution $p(x|\theta)$. To give a concrete example, assume that θ models the space of human faces: then, given fixed $\theta := \theta_0$, the probability of observing face #1 over face #2 could be:

$$p(x_1|\theta = \theta_0) > p(x_2|\theta = \theta_0),$$

while, for a different θ realization, $\theta := \theta_1$, it might be:

$$p(x_1|\theta = \theta_1) < p(x_2|\theta = \theta_1).$$

Now, assume that we have a data set $\{x_i\}_{i=1}^n$. One way to learn θ is through maximum log-likelihood: we define the log-likelihood as $\log p(x|\theta)$, and we are interested in:

$$\widehat{\theta} \in \arg\max_{\theta \in \mathbb{R}^p} \left\{\mathcal{L}(\theta) := \mathbb{E}_{p(x|\theta)}\left[\log p(x|\theta)\right]\right\}$$

Let us compute the gradient and the Hessian of this new function. For the gradient, we first compute:

$$\nabla \log p(x|\theta) = \tfrac{1}{p(x|\theta)} \cdot \nabla p(x|\theta)$$

and thus,

$$\nabla \mathcal{L}(\theta) = \mathbb{E}_{p(x|\theta)}\left[\tfrac{1}{p(x|\theta)} \cdot \nabla p(x|\theta)\right].$$

For the Hessian, as the Jacobian of the gradient, we have:

$$\begin{aligned} H_{\log p(x|\theta)} &= \nabla\left(\tfrac{1}{p(x|\theta)} \cdot \nabla p(x|\theta)\right) \\ &= \tfrac{H_{p(x|\theta)} \cdot p(x|\theta) - \nabla p(x|\theta) \cdot \nabla p(x|\theta)^\top}{p(x|\theta) \cdot p(x|\theta)} \\ &= \frac{H_{p(x|\theta)}}{p(x|\theta)} - \left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right) \cdot \left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right)^\top \end{aligned}$$

where $H_{p(x|\theta)}$ is the Hessian with respect to $p(x|\theta)$. Computing the expectation with respect to $p(x|\theta)$, we have:

$$\begin{aligned} &\mathbb{E}_{p(x|\theta)}\left[H_{\log p(x|\theta)}\right] \\ &= \mathbb{E}_{p(x|\theta)}\left[\frac{H_{p(x|\theta)}}{p(x|\theta)}\right] - \mathbb{E}_{p(x|\theta)}\left[\left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right) \cdot \left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right)^\top\right] \\ &= \int \frac{H_{p(x|\theta)}}{p(x|\theta)} p(x|\theta)dx - \mathbb{E}_{p(x|\theta)}\left[\left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right) \cdot \left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right)^\top\right] \end{aligned}$$

Before we proceed, observe that:

$$\begin{aligned} \int \frac{H_{p(x|\theta)}}{p(x|\theta)} p(x|\theta)dx &\equiv \int \nabla\left(\tfrac{1}{p(x|\theta)} \cdot \nabla p(x|\theta)\right) \cdot p(x|\theta)dx \\ &= \int \nabla\left(\tfrac{1}{p(x|\theta)} \cdot \nabla p(x|\theta) \cdot p(x|\theta)\right) dx \\ &= \int \nabla^2\left(p(x|\theta)\right) dx \\ &\overset{\text{(mild assumptions)}}{=} \nabla^2\left(\int p(x|\theta)dx\right) = \nabla^2(1) = 0 \end{aligned}$$

Thus,

$$\mathbb{E}_{p(x|\theta)}\left[H_{\log p(x|\theta)}\right] = -\mathbb{E}_{p(x|\theta)}\left[\left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right) \cdot \left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right)^\top\right]$$

where the quantity on the right hand side is the *Fisher information*, usually denoted with F. We can see the Fisher information as a measure of curvature for the log-likelihood function.

But how are we going to use this information in optimization? One can think of the immediate application of F as a replacement of Hessian in second order methods. Nevertheless, this is not obvious why, and under which settings.

One key difference so far in our narrative is the introduction of a probability distribution $p(x|\theta)$. In our course thus far, we have discussed about *deterministic optimization*: We are just given an objective $f(x)$, that is usually differentiable, and we try to find the minimum/maximum. However, "under the rag", we have implied that any step we will perform, it will be measured in the Euclidean space. To see this, assume that $\mathcal{L}(\theta)$ denotes the negative log-likelihood that we want to minimize. Gradient descent is one way to do so: we compute the direction d on the parameter space θ that minimizes the objective. Formally, we can find that the best direction is actually the negative gradient:

$$\lim_{\epsilon \to 0}\left(\tfrac{1}{\epsilon} \arg\min_{\|d\| \leq \epsilon} \mathcal{L}(\theta + d)\right) = -\frac{\nabla \mathcal{L}(\theta)}{\|\nabla \mathcal{L}(\theta)\|_2}.$$

i.e., the direction (*that is why we have normalization; we care about the direction, not how far we go on this direction*) of minimum drop on $\mathcal{L}(\cdot)$ is the negative gradient.

By definition of this steepest descent direction though, we use the Euclidean norm. Thus, the optimization in gradient descent is dependent on the Euclidean geometry of the parameter space.

Though, we have introduced the notion of likelihoods and expectations; stated differently, we have different objectives by minimizing the negative log-likelihood loss function, and it is natural to think of steps in the space of all possible likelihood, realizable by parameter θ.

The Kullback-Leibler divergence. First, we need to define the notion of the Kullback-Leibler (KL) divergence metric. Let $p_1(\cdot)$, $p_2(\cdot)$ be two distributions. Then, the KL divergence is given by:

$$D_{\text{KL}}\left(p_1(\cdot)||p_2(\cdot)\right) = \mathbb{E}_{p_1(\cdot)}\left[\log \tfrac{p_1(\cdot)}{p_2(\cdot)}\right].$$

The KL-divergence measure the "closeness" of two distributions. To give an example: consider two Gaussians, with their means fixed in the two plots to follow (Figure 31; borrowed from `https://wiseodd.github.io/techblog/2018/03/14/natural-gradient`), but with different variances.

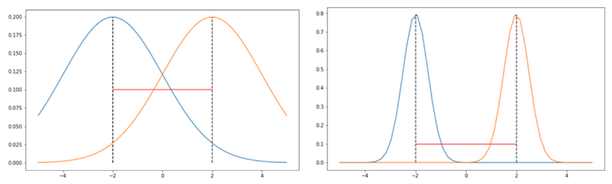

Fig. 31. Graphical illustration of distance metrics between distributions.

If we were to use the Euclidean distance between the means as a metric between the two distributions, it is clear that the metric would be the same between the two cases: since the means remain the same, their ℓ_2-norm distance is the same.

Nevertheless, the distributions in these two plots are different, and thus we should find a metric that mirrors this in its definition.

Properties of the KL divergence: Observe that the KL divergence, by definition, satisfies:

$$D_{\text{KL}}\left(p_1(\cdot)||p_2(\cdot)\right) = \mathbb{E}_{p_1(\cdot)}\left[\log \tfrac{p_1(\cdot)}{p_2(\cdot)}\right] = \mathbb{E}_{p_1(\cdot)}\left[\log p_1(\cdot)\right] - \mathbb{E}_{p_1(\cdot)}\left[\log p_2(\cdot)\right].$$

To help our discussion, and make connections with our problem so far, we use $p_1(\cdot) = p(x|\theta)$ and $p_2(\cdot) = p(x|\theta')$. Then, the gradient of KL with respect to θ' satisfies:

$$\begin{aligned}&\nabla_{\theta'}\left(D_{\text{KL}}\left(p(x|\theta)||p(x|\theta')\right)\right)\\ &= \nabla_{\theta'}\mathbb{E}_{p(x|\theta)}[\log p(x|\theta)] - \nabla_{\theta'}\mathbb{E}_{p(x|\theta)}[\log p(x|\theta')]\\ &= -\mathbb{E}_{p(x|\theta)}[\nabla_{\theta'}\log p(x|\theta')]\\ &= -\int p(x|\theta)\nabla_{\theta'}\log p(x|\theta')\,\mathrm{d}x.\end{aligned}$$

and the second derivative satisfies:

$$\nabla^2_{\theta'}\left(D_{\text{KL}}\left(p(x|\theta)||p(x|\theta')\right)\right) = -\int p(x|\theta)\,\nabla^2_{\theta'}\log p(x|\theta')\,\mathrm{d}x$$

Then, the Hessian evaluated at $\theta' = \theta$ is:

$$\begin{aligned}\mathrm{H}_{D_{\text{KL}}(p(x|\theta)||p(x|\theta'))} &= -\int p(x|\theta)\ \nabla^2_{\theta'}\log p(x|\theta')\big|_{\theta'=\theta}\,\mathrm{d}x\\ &= -\int p(x|\theta)\,\mathrm{H}_{\log p(x|\theta)}\,\mathrm{d}x\\ &= -\underset{p(x|\theta)}{\mathbb{E}}\left[\mathrm{H}_{\log p(x|\theta)}\right]\\ &= F,\end{aligned}$$

which is what we have shown above; i.e., the expected Hessian of the log function is the Fisher information matrix.

2nd-oder Taylor expansion of KL divergence and natural gradient: Let us now connect the dots. Following similar reasoning to classical optimization, given an objective function, we can approximate locally the objective with its second-order Taylor approximation (which involves both the gradient and the Hessian information), and then locally minimize that approximation; then, we iterate.

The second-order approximation of the KL divergence metric satisfies:

$$\begin{aligned}&D_{\text{KL}}\left(p(x|\theta)||p(x|\theta+d)\right)\\ &\approx D_{\text{KL}}\left(p(x|\theta)||p(x|\theta)\right) + \left\langle\nabla D_{\text{KL}}\left(p(x|\theta)||p(x|\theta)\right), d\right\rangle + \frac{1}{2}\langle Fd, d\rangle\\ &= \frac{1}{2}\langle Fd, d\rangle.\end{aligned}$$

Similar to the Euclidean case, we seek for an update vector d that minimizes the loss function $\mathcal{L}(\theta)$ in the *distribution space*. Analogously to steepest descent:

$$d^\star = \text{argmin}_{D_{\text{KL}}(p(x|\theta)||p(x|\theta+d))=c}\mathcal{L}(\theta+d),$$

where c is some constant. Compare this with the Euclidean case where:

$$d^\star = \text{argmin}_{\|d\|_2\leq\epsilon}\mathcal{L}(\theta+d).$$

The purpose of fixing the KL-divergence to some constant is to make sure that we move along the space of distributions with constant speed, regardless the curvature.

How do we solve this part? If we write the above minimization in Lagrangian form (*we have not talked about Lagrange multipliers yet, but accept this for the moment*), we get:

$$\begin{aligned}d^\star &= \arg\min_d\left\{\mathcal{L}(\theta+d) + \lambda\cdot\left(D_{\text{KL}}\left(p(x|\theta)||p(x|\theta+d)\right) - c\right)\right\}\\ &\approx \arg\min_d\left\{\mathcal{L}(\theta) + \langle\nabla\mathcal{L}(\theta),\ d\rangle + \frac{1}{2}\lambda\langle Fd,\ d\rangle - \lambda c\right\}.\end{aligned}$$

To solve this minimization, we set its derivative with respect to d to zero; this will lead to the solution:

$$d = -\tfrac{1}{\lambda}F^{-1}\nabla\mathcal{L}(\theta).$$

(The λ constant can be absorbed in the definition of F.)

The above lead to the definition of the *natural gradient* descent method:

- **Repeat:**
  1. Compute the gradient $\nabla\mathcal{L}(\theta_t)$.
  2. Compute the Fisher information matrix F_t.
  3. Compute the natural gradient direction: $d_t = F_t^{-1}\nabla\mathcal{L}(\theta_t)$.
  4. For a step size η, compute $\theta_{t+1} = \theta_t - \eta d_t$.

Take-away messages:

- The natural gradient descent is a generalization of the Newton's method, as there are cases where from the natural gradient descent we can obtain the Newton's iteration.
- Remember that, afterall, the Fisher information matrix is computed per iteration, and inverted; and it turns out that on expectation it corresponds to the expected Hessian of the objective.
- How do we implement the natural gradient method in reality? Remember the definition of the Fisher information:

$$F = -\underset{p(x|\theta)}{\mathbb{E}}\left[H_{\log p(x|\theta)}\right].$$

In realistic scenarios, we do not have access to the distribution $p(x|\theta)$, but rather have data that come from that distribution. In that case, we refer to the *empirical* Fisher information matrix, defined as:

$$F = \tfrac{1}{n}\sum_{i=1}^n \nabla\log p(x_i|\theta)\cdot\nabla\log p(x_i|\theta)^\top$$

where $\{x_i\}_{i=1}^n$ denote the training set of examples. In that case, the main recursion of natural gradient descent becomes:

$$\theta_{t+1} = \theta_t - \eta\left(\tfrac{1}{n}\sum_{i=1}^n\nabla\log p(x_i|\theta_t)\cdot\nabla\log p(x_i|\theta_t)^\top\right)^{-1}\cdot\nabla\widehat{\mathcal{L}}(\theta_t),$$

where also the objective and its gradient are evaluated in their empirical form:

$$\nabla\widehat{\mathcal{L}}(\theta_t) := \tfrac{1}{n}\sum_{i=1}^n\nabla\log p(x_i|\theta_t).$$

- The main reason we studied natural gradient descent is to motivate our discussion later on, regarding algorithms in training neural networks. The empirical Fisher information matrix appears in almost all modern algorithms in ML, usually further approximated to be easily computed. E.g., one way to get around computing the exact empirical Fisher information matrix is to constrain it to be diagonal matrix; a technique that is heavily used in algorithms such as AdaGrad, AdaDelta, RMSprop, Adam, AMSGrad, Yogi, etc. In other words, in the mostly used algorithms in neural network training.