5

Unconstrained Optimization

In unconstrained optimization we want to find the global minimum of a function $f : \mathbb{R}^n \to \mathbb{R}$. Without further assumptions, this problem is undecidable, so in order to be able to design (and analyze) algorithms, we need to restrict the class of functions we can tackle.

In the computer science meaning of the word: there is no algorithm with guaranteed termination.

For starters, we make two basic assumptions:

1. the function f admits an optimal solution x^* of value p^*.

   This also means that f is bounded below by p^*.

2. the gradient ∇f cannot change arbitrarily fast. More formally, we assume that ∇f is Lipschitz continuous, i.e.:

$$\|\nabla f(x) - \nabla f(y)\| \leq M\|x - y\| \tag{1}$$

   for some fixed constant $M \geq 0$ in the domain of the function. For $\mathbb{C}^2$ functions, this is equivalent to $\nabla^2 f(x) \preceq MI$.

Since the function is sufficiently smooth (at least C^1 given the assumptions above), the first order conditions for optimality require that $\nabla f(x^*) = 0$ on any optimal solution x^*. So, at least in principle, a first step is to solve such system of n *nonlinear* equations in n variables. Unlike linear systems, this in general cannot be solved analytically, so we resort to iterative methods that compute a sequence of points:

For convex functions, this is also a sufficient *condition for global optimality!*

$$x^0, x^1, x^2, \ldots$$

converging to a solution satisfying the first order condition to the limit:

$$\lim_{k \to +\infty} \nabla f(x^k) = 0$$

In the following, we will only consider iterative methods that maintain primal feasibility, i.e., $x^k \in \operatorname{dom} f$ for all k. Also, we will only consider *descent* methods, i.e., iterative schemes where $f(x^{k+1}) < f(x^k)$. Note that, given a feasible starting point x^0, we can without loss of generality restrict our search to the sublevel set $S = \{x \in \operatorname{dom} f | f(x) \leq f(x^0)\}$: we will often assume that this set S is compact, i.e., closed and bounded, as this vastly increases the set of functions for which condition (1) is satisfied–it just need to hold for the points in S.

This means that the objective function monotonically improves at every step.

5.1 Iterative methods

Without loss of generality, all iterative methods share the same pattern:

$$x^{k+1} = x^k + t^k \Delta x^k$$

This is indeed referred to as step length *or* step size.

In other words, the next point x^{k+1} in the sequence is obtained by the current point x^k and taking a step of length $t^k \geq 0$ along the *search direction* Δx^k. The algorithm will start from some initial point x^0, and continue until some stopping criterion is met, usually based on the norm the gradient, e.g., $\|\nabla f(x^k)\| \leq \varepsilon$. A pseudocode is available in Algorithm 3.

Algorithm 3: Generic iterative scheme.

```
let x ∈ dom f be a starting point
while ‖∇f(x)‖ > ε do
    Determine direction Δx
    Choose step size t
    Update x ← x + tΔx
end
return x
```

Of course, different choices of the search direction and/or stepsize give rise to different algorithms, with different convergence properties.

5.1.1 *Choice of step length*

Suppose the search direction Δx is already chosen: how do we pick a step size? We have two conflicting goals here: on the one hand, the direction chosen (e.g., the gradient) might give a valid approximation of the function only locally, so we cannot take too large of a step; on the other hand, we also don't want the step to be too small when a larger one could have been beneficial, as that can slow down convergence significantly.

There are basically two options:

Remember, x and Δx are fixed.

1. *exact line search*: solves the one-dimensional optimization problem

$$t = \arg\min_{s \geq 0} f(x + s\Delta x)$$

   This is used when t can be computed analytically and/or its costs is low compared to finding Δx.

2. *line search* (inexact): does not look for the optimal t, just aims at reducing f *enough*.

A very simple, yet effective, strategy of line search is called *backtracking* line search, based on the Armijo-Goldstein conditions. It is parametrized by two constants, $0 < \alpha < \frac{1}{2}$ and $0 < \beta < 1$, and it consists in starting with a step length $t = 1$, and then reducing it geometrically, with a factor β, until the stopping condition $f(x + t\Delta x) \leq f(x) + \alpha t \nabla f(x)^\top \Delta x$ is met. A pseudocode is available in Algorithm 4.

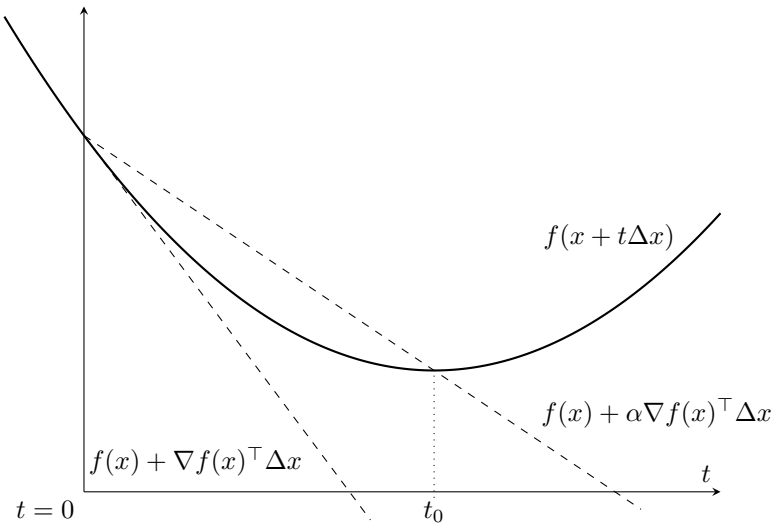


Figure 5.1: Backtracking line search.

Algorithm 4: Backtracking line search.

let $t = 1$
while $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^\top \Delta x$ **do**
  $t \leftarrow \beta t$
end
return t

A graphical representation of the stopping condition is depicted in Figure 5.1. The idea is that we decrease the slope of the tangent at $t = 0$ by a factor α, and look for the largest value of t such that f is below the modified line. If t_0 is indeed the largest such value, it can be shown that when line search stops we either have $t = 1$ (the initial step already satisfies the condition) or $t \in (\beta t_0, t_0]$ (we are at most a factor of β away from t_0). Either way, we have that $t \geq \min\{1, \beta t_0\}$, so t cannot be too small.

5.2 Gradient Descent

How do we pick a search direction? If the function to optimize is convex, then by convexity alone we have:

$$f(x^{k+1}) \geq f(x^k) + \nabla f(x^k)^\top \underbrace{(x^{k+1} - x^k)}_{t^k \Delta x^k}$$

so if we want a descent direction it must hold that:

$$\nabla f(x^k)^\top \Delta x^k < 0 \tag{2}$$

The most obvious choice is $\Delta x = -\nabla f(x)$, so the negative gradient. It clearly satisfies (2), as we get

$$\nabla f(x^k)^\top \Delta x^k = -\|\nabla f(x^k)\|^2 < 0$$

unless the gradient is null, in which case we are done.

More on this later.

Even if the function is non-convex, choosing the negative gradient is a very natural choice, as the negative gradient is the direction of steepest descent w.r.t. the local first order approximation of the function.

So the question becomes: does such a method converge? If it does, at which speed? Note that, in all those schemes, the overall complexity is the product of two terms: the *iteration cost*, i.e., how expensive each individual iteration is, and the *iteration complexity*, i.e., how many iterations are needed in order to reach a desired accuracy.

5.2.1 *Descent Lemma*

The condition (1) alone implies the so-called *descent* lemma:

$$f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{M}{2}\|y - x\|^2 \tag{3}$$

Proof.

$$\begin{aligned}
f(y) &= f(x) + \int_0^1 \nabla f(x + a(y - x))^\top (y - x) da && \text{(fund. thm. calculus)} \\
&= f(x) + \nabla f(x)^\top (y - x) + \int_0^1 (\nabla f(x + a(y - x)) - \nabla f(x))^\top (y - x) da && (\pm \text{ constant}) \\
&\leq f(x) + \nabla f(x)^\top (y - x) + \int_0^1 \|\nabla f(x + a(y - x)) - \nabla f(x))\|\|(y - x)\| da && \text{(Cauchy-Schwarz)} \\
&\leq f(x) + \nabla f(x)^\top (y - x) + \int_0^1 M\|x + a(y - x) - x\|\|(y - x)\| da && \text{(Lipschitz)} \\
&= f(x) + \nabla f(x)^\top (y - x) + \int_0^1 aM\|y - x\|^2 da && \text{(factor out)} \\
&= f(x) + \nabla f(x)^\top (y - x) + \frac{1}{2} M\|y - x\|^2
\end{aligned}$$

□

The descent lemma implies that we have a quadratic upper bound on our function f, which is minimized by:

$$\hat{y} = x - \frac{1}{M}\nabla f(x)$$

A much simpler strategy than backtracking line search or exact methods.

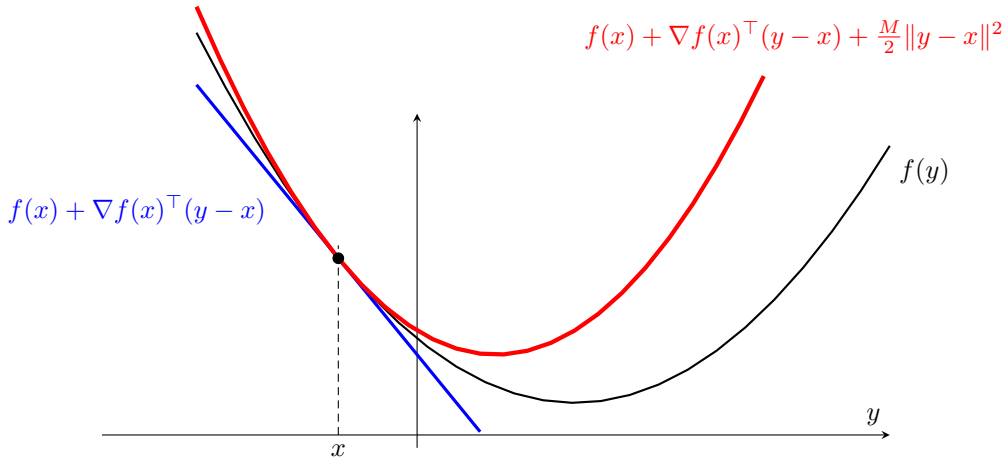
This means that gradient descent with a constant step

$$t^k = \frac{1}{M}$$

will always achieve some improvement that is enough for convergence (albeit a slow one).

Proof. With a step length of $\frac{1}{M}$ and a direction of $-\nabla f(x^k)$, the difference $x^{k+1} - x^k$ is equal to $-\frac{1}{M}\nabla f(x^k)$. Plugging this into the inequality of the descent lemma we get:

$$\begin{aligned}
f(x^{k+1}) &\leq f(x^k) + \nabla f(x^k)^\top (x^{k+1} - x^k) + \frac{M}{2}\|x^{k+1} - x^k\|^2 \\
&= f(x^k) - \frac{1}{M}\|\nabla f(x^k)\|^2 + \frac{M}{2}\frac{1}{M^2}\|\nabla f(x^k)\|^2 \\
&= f(x^k) - \frac{1}{2M}\|\nabla f(x^k)\|^2
\end{aligned}$$



so we have a guaranteed progress of $\frac{1}{2M}\|\nabla f(x^k)\|^2$.

This is enough to get our first convergence result.

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2M}\|\nabla f(x^k)\|^2$$
$$\|\nabla f(x^k)\|^2 \leq 2M(f(x^k) - f(x^{k+1}))$$

Now we can just average the last inequalities over the first s terms:

$$\underbrace{\frac{1}{s}\sum_{k=0}^{s-1}\|\nabla f(x^k)\|^2}_{\text{minimum gives lower bound}} \leq \frac{2M}{s}\underbrace{\sum_{k=0}^{s-1}(f(x^k) - f(x^{k+1}))}_{\text{telescoping sum}}$$

and obtain

$$\min_{k=0,\ldots,s-1}\|\nabla f(x^k)\|^2 \leq \frac{2M}{s}(f(x^0) - f(x^s))$$
$$\leq \frac{2M}{s}(f(x^0) - p^*)$$

So in s iterations we find at least one iteration where

$$\|\nabla f\|^2 = O(\frac{1}{s})$$

This is not necessarily the last iteration, but this is enough to show that

$$\liminf_{s\to+\infty}\|\nabla f\|^2 = 0$$

□

What does this imply for accuracy? We have that:

$$\frac{2M(f(x^0) - p^*)}{s} \leq \varepsilon$$

and thus

$$s \geq \frac{2M(f(x^0) - p^*)}{\varepsilon}$$

so $s = O(\frac{1}{\varepsilon})$. This is a *bad* iteration complexity, as it is exponential in the number of digits of accuracy, i.e., $\log(\frac{1}{\varepsilon})$. Note that the convergence above is obtained with a constant step size, but it applies to exact line search as well, as that can only make a larger progress at each step.

Note that in practice we do not know M, but we can guess it. Start with $M = 1$ and double it until

$$f\left(x^k - \frac{\nabla f(x^k)}{M}\right) \leq f(x^k) - \frac{\|\nabla f(x^k)\|^2}{2M}$$

In practice this is even better than knowing the real M, but the convergence is very slow. Backtracking line search is way better!

Note that we didn't even assume convexity! For non-convex functions, we will just converge to a stationary point.

5.2.2 *Polyak-Łojasiewicz Inequality*

The iteration complexity if $O(\frac{1}{\varepsilon})$ is the best we can prove without further assumptions on the function f we are optimizing. We can achieve a much better complexity if we assume the so-called Polyak-Łojasiewicz (PL) inequality:

$$\frac{1}{2}\|\nabla f(x)\|^2 \geq m(f(x) - p^*)$$

for all $x \in S$ and some $m > 0$.

If f is not $\mathbb{C}^2$, strong convexity can be defined as $f(x) - \frac{m}{2}\|x\|^2$ still convex for some $m > 0$, and we still have that strong convexity implies the PL inequality.

For functions of class $\mathbb{C}^2$, strong convexity implies the PL inequality, with the same constant m:

$$\nabla^2 f(x) \succeq mI$$

Assuming the PL inequality, we can prove a much better iteration complexity for the same gradient scheme with constant step size $t^k = \frac{1}{M}$. Starting again from:

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2M}\|\nabla f(x^k)\|^2$$

we can now exploit that

$$-\|\nabla f(x^k)\|^2 \leq -2m(f(x^k) - p^*)$$

and thus, combining the two:

$$f(x^{k+1}) - p^* \leq f(x^k) - p^* - \frac{m}{M}(f(x^k) - p^*)$$

It is easy to show that $m \leq M$.

which directly implies that at each iteration the error decreases at least by a factor of $(1 - \frac{m}{M}) < 1$. Iterating over s steps, we finally get:

$$f(x^s) - p^* \leq \left(1 - \frac{m}{M}\right)^s (f(x^0) - p^*)$$

And linear in the number of digits of accuracy

so $f(x^s) - p^* = O(\rho^s)$ for some $\rho = (1 - m/M) < 1$. This is called exponential convergence, and it implies that to achieve an error of ε we need $O(\log \frac{1}{\varepsilon})$ iterations. Indeed, if we impose that the error after s iterations is bounded by ε and rearrange we get:

Again, this applies also to exact line search and, with minimal adjustments, to backtracking line search as well.

$$s \geq \frac{\log \frac{(f(x^0) - p^*)}{\varepsilon}}{\log \frac{1}{\rho}} = O\left(\log \frac{1}{\varepsilon}\right) \tag{4}$$

Note that in (4), the numerator depends on how far from the optimal value the initial point x^0 is, relative to the desired tolerance ε, while the denominator depends on the constants m and M and thus, ultimately, on the structure of the function: if the so called condition number M/m is large, then ρ is very close to 1 and convergence is very slow. On the other hand, if the condition number is small (close to 1), then ρ is close to zero and convergence is very fast.

Practical considerations. The analysis above is verified in practice in computational evaluations, and it turns out to be quite accurate. In particular:

- the effect of α and β within backtracking line search is noticeable but not dramatic;
- the same can be stated about the difference between exact line search and backtracking line search;

- convergence is truly sensitive to the condition number M/m, and it becomes painfully slow if it this number is even modestly large (say, ≥ 1000).

Note that the iteration complexity does not depend directly on the dimension n, although the constants m and M might. Also, note that in practice m and M are unknown: they are just conceptual tools used in the convergence analysis, to understand the behaviour of the method.

So, we can conclude that the main advantage of gradient decent is that it is very simple and the cost of each iteration is small, but at the cost of having an iteration complexity which is too sensitive to the condition number, and potential very slow if this number is even modestly large.

5.3 Stochastic Gradient Descent

There are many applications where the objective function to be minimized $f(x)$ can be expressed as a sum (or average) of functions over a given set D:

$$f(x) = \frac{1}{|D|} \sum_{i \in D} f_i(x)$$

The most important example is machine learning (ML), where D is the set of samples/observations/data points. In this case, computing the gradient, while still technically linear in the dimension n, can be prohibitively expensive, as it also depends linearly on $d = |D|$, and we usually have $d \gg n$.

Its complexity is $O(nd)$.

In those cases, gradient descent is modified to replace the gradient with a cheaper approximation, that can be obtained by only using a (small) subset of the functions in the set D at each iteration (the so-called *sampled gradient*): this variant is called *stochastic gradient descent* (SGD), and it is one of the backbones of modern ML.

While quite natural and intuitive, the sample gradient is *not* a gradient of $f(x)$, so monotonic decrease is not guaranteed–and indeed does not happen in practice. Can we still prove convergence of the overall scheme? At which rate?

Our analysis is based on the simplest variant of SGD, where at each iteration we pick only a single sample from D. This is called *online SGD*. The key observation for proving convergence is that the sampled gradient is an *unbiased* estimate of the true gradient, assuming the sample is drawn *uniformly at random*. More formally:

$$\begin{aligned} E\left[\nabla f_{i_k}(x)\right] &= \sum_{i \in D} p[i_k = i] \nabla f_i(x) \\ &= \frac{1}{|D|} \sum_{i \in D} \nabla f_i(x) \\ &= \nabla f(x) \end{aligned}$$

However, we will see that in this case we will need the step size to converge to zero (there is no convergence with a constant step size).

Proof. Let's start again with the descent lemma:

$$f(x^{k+1}) \leq f(x^k) + \nabla f(x^k)^\top (x^{k+1} - x^k) + \frac{M}{2} \|x^{k+1} - x^k\|^2$$

and plug in $x^{k+1} - x^k = -t_k \nabla f_{i_k}(x^k)$. We obtain:

$$f(x^{k+1}) \leq f(x^k) - t_k \nabla f(x^k) \nabla f_{i_k}(x^k) + t_k^2 \frac{M}{2} \|\nabla f_{i_k}(x^k)\|^2$$

We can now take the expectation on both sides, assuming again uniform sampling:

$$E\left[f(x^{k+1})\right] \leq f(x^k) - t_k \nabla f(x^k) \underbrace{E\left[\nabla f_{i_k}(x^k)\right]}_{\nabla f(x^k)} + t_k^2 \frac{M}{2} E\left[\|\nabla f_{i_k}(x^k)\|^2\right]$$

$$= f(x^k) - t_k \|\nabla f(x^k)\|^2 + t_k^2 \frac{M}{2} E\left[\|\nabla f_{i_k}(x^k)\|^2\right]$$

The last term is the expression above is *bad*:

- it reduces progress if the gradients are different;
- it makes it very risky to take large steps.

On the the other hand, notice that if t_k is small, then $t_k \gg t_k^2$. For the sake of simplicity, let us make the *finite variance* assumption:

$$E\left[\|\nabla f_{i_k}(x^k)\|^2\right] \leq \sigma^2$$

for some constant σ^2. Then the expression simplifies to

$$E\left[f(x^{k+1})\right] \leq f(x^k) - t_k \|\nabla f(x^k)\|^2 + t_k^2 \frac{M}{2} \sigma^2$$

and, rearranging:

$$t_k \|\nabla f(x^k)\|^2 \leq f(x^k) - E\left[f(x^{k+1})\right] + t_k^2 \frac{M}{2} \sigma^2$$

At this point we can sum up the first $s+1$ terms, and exploit again the usual tricks (take expectation, bound by minimum, and telescoping sums):

$$\sum_{k=0}^{s} t_k \|\nabla f(x^k)\|^2 \leq \sum_{k=0}^{s} \left(f(x^k) - E\left[f(x^{k+1})\right]\right) + \frac{M}{2}\sigma^2 \sum_{k=0}^{s} t_k^2$$

$$\sum_{k=0}^{s} t_k \underbrace{E\left[\|\nabla f(x^k)\|^2\right]}_{\text{lower bounded by min}} \leq \sum_{k=0}^{s} \underbrace{\left(E\left[f(x^k)\right] - E\left[f(x^{k+1})\right]\right)}_{\text{telescoping sum}} + \frac{M}{2}\sigma^2 \sum_{k=0}^{s} t_k^2$$

$$\min_{k=0,\ldots,s} E\left[\|\nabla f(x^k)\|^2\right] \sum_{k=0}^{s} t_k \leq f(x^0) - \underbrace{E[f(x^s)]}_{\geq p^*} + \frac{M}{2}\sigma^2 \sum_{k=0}^{s} t_k^2$$

Finally, dividing by $\sum_{k=0}^{s} t_k$, we get

$$\min_{k=0,\ldots,s} E\left[\|\nabla f(x^k)\|^2\right] \leq \frac{f(x^0) - p^*}{\sum_{k=0}^{s} t_k} + \frac{M}{2}\sigma^2 \frac{\sum_{k=0}^{s} t_k^2}{\sum_{k=0}^{s} t_k}$$

So, in order for the (expected) gradient to converge to zero, we need the two conditions:

$$\sum_k t_k \to +\infty \quad \sum_k t_k^2 \to 0$$

□

For example, with $t_k = a/k$ for some constant a, we have that

$$\sum_{k=1}^{s} t_k = O(\log s) \quad \sum_{k=1}^{s} t_k^2 = O(1)$$

so the error after s iterations is $O(1/\log s)$, which is worse than standard gradient descent. If, instead, we pick a constant step size $t_k = a$, we get an error of $O(1/s) + O(a)$, that does *not* converge to zero as $k \to +\infty$.

For strongly convex functions, a similar analysis gives, with the decreasing step size rule $t_k = 1/mk$, an error of $O(1/s)$ after s iterations,, while with a constant step size $t_k = a < 1/2m$ we get that an error bound of $O(\rho^s) + O(a)$, which means linear convergence but only up to some accuracy proportional to a. Note that for both strongly convex and general functions, using a constant step size gives fast convergence at the beginning, but eventually the second (constant) term dominates: this justifies the heuristic used in practice of *"halving the step size if the process is not making progress"*, as the second term depends on a.

Again, worse than in the standard case.

5.3.1 *Mini-batch SGD*

As we have seen, the standard gradient descent achieves faster convergence than SGD, but at the cost of d gradient evaluations per iteration, instead of just 1. Is there a middle ground between the two? A very natural approach consists in using more samples at each iteration, in order to have a more accurate approximation of the true gradient at x^k: this is called the *mini-batch* approach, and it consists in randomly sampling a subset B^k of indices at each iteration.

Also called deterministic.

On modern parallel architectures, if the size of B^k is appropriately chosen, we can evaluate those gradients equally fast, so the iteration cost is not even higher.

The k-th iteration of the method can thus be written as:

$$x^{k+1} = x^k - t_k \frac{1}{|B^k|} \sum_{i \in B^k} \nabla f_i(x^k)$$

Let us consider the mini-batch direction as the true gradient $\nabla f(x^k)$ plus some noise term e^k that depends on the sample we have chosen. By using a constant step size $t_k = 1/M$, and repeating the steps of the descent lemma, we get the inequality:

$$f(x^{k+1}) \leq f(x^k) - \underbrace{\frac{1}{2M}\|\nabla f(x^k)\|^2}_{\text{good term}} + \underbrace{\frac{1}{2M}\|e^k\|^2}_{\text{bad term}}$$

As in the online case, the error term is usually the limiting factor for convergence. How does $|B^k|$ affect $\|e^k\|^2$? Under the usual finite variance assumption, it is easy to show that if we are sampling *with replacement*, we have:

$$E[\|e^k\|^2] = \frac{1}{|B^k|}\sigma^2$$

and thus, unsurprisingly, the error gets smaller as we increase the size of the mini-batch. Even better, if we sample *without replacement* from the full set of d gradients, we obtain:

$$E[\|e^k\|^2] = \frac{d - |B^k|}{d} \frac{1}{|B^k|}\sigma^2$$

and this goes to zero as $|B^k| \to d$. In other words, if we increase the size of the mini-batch over time at the appropriate rate, then can reduce the error

$\|e^k\|^2$ fast enough that it is no longer the limiting factor, and we recover the convergence of standard gradient descent. For example, with the rule $|B^{k+1}| = |B^k|/\rho$ we can achieve $O(\rho^s)$ convergence, while with $|B^{k+1}| = |B^k| + a$ we match $O(1/s)$. Note, however, that those update rules are not used in implementations, as they increase the mini-batch size too quickly: in practice we prefer to keep the mini-batch size fixed (and carefully chosen to match the available hardware), and double its size only when stalling is detected.