Chapter 7

In the last chapter, we considered natural ways of accelerating the performance of gradient descent, by cleverly using previous estimates via the momentum term. However, while acceleration in that setting leads to faster convergence in analytical complexity terms, it does not reduce the per iteration complexity.

In this chapter, we will discuss how we can accelerate the performance of gradient methods by computing less per iteration: based on the empirical risk minimization formulation, we will discuss stochasticity and how the stochastic version of gradient descent can be beneficial in practice. Definitely, such a choice creates trade-offs (number of iterations to converge vs. computational complexity per iteration). As an alternative way of accelerating gradient descent, we also mention the notion of coordinate descent, where we update individual entries of the variable vector, instead of the full vector, per iteration (more details in class).

Stochastic gradient descent and its variants | Coordinate descent and its variants

To discuss about stochasticity in optimization, we will first need a new problem definition, that is amenable to stochasticity. While stochastic optimization and stochastic approximations are large research areas by themselves, a particular optimization criterion that appears often in the literature is that of the empirical risk minimization. Let us define that through an example.

Consider that we have a dataset $\mathcal{D}$ that is comprised by n data points $\{w_i, y_i\}_{i=1}^n$:[11] here, we can simplify the discussion by assuming that w_i is a vectorized version of an image, $w_i \in \mathbb{R}^m$, and $y_i \in \mathbb{R}$ is the label for that image. We are interested in designing a predictor function $h : \mathbb{R}^m \rightarrow \mathbb{R}$ that takes as an input an image w_i and predicts its correct label y_i. This predictor is to be learned, and the learning parameters are described via a real vector $x \in \mathbb{R}^p$, such that $h(x, w_i) = y_i$. To make this even more explicit, if we were talking specifically about the case of linear regression, and under the setting $p = m$, the predictor should be:

$$h(x, w_i) = x^\top w_i = y_i, \forall w_i.$$

In the above description, what is vague is the statement "$\forall w_i$". What if we have $n = 1000$ samples and then we have 1000 more? Should we create a predictor that operates flawlessly on the given dataset, but gives no guarantees how it operates on unseen data? This leads to the notion of *expected risk*: can we find parameters x that minimize the prediction loss on average, over all possible input-output data points? I.e., if we measure the loss as $(h(w, x) - y)^2$, can we optimize:

$$\min_x \mathbb{E}_{\{x,y\}\sim\mathcal{D}} \left[(h(w, x) - y)^2\right].$$

In reality, we do not have access to the true distribution of how data is generated. Thus, we cannot practically optimize this objective, but we can approximate it through the *empirical risk*: assuming that each data point is generated i.i.d. in a uniform way, we can approximate the expected risk with the empirical risk, and solve:

$$\min_x \tfrac{1}{n} \sum_{i=1}^n \left[(h(w_i, x) - y_i)^2\right].$$

Let us define $f_i(x) := (h(w_i, x) - y_i)^2$. Then, the above objective is written as:

$$\min_x \left\{ f(x) := \tfrac{1}{n} \sum_{i=1}^n f_i(x) \right\}.$$

How would we solve this problem? By applying gradient descent methods! In particular, we can compute the gradient at the iteration t as:

$$\nabla f(x_t) = \nabla \left[\sum_{i=1}^n f_i(x_t)\right] = \sum_{i=1}^n \nabla f_i(x_t);$$

i.e., the full gradient is actually the summation of the gradients of each individual $f_i(\cdot)$. Thus, if the computation of $\nabla f_i(\cdot)$ takes $\mathtt{T}_{\nabla f_i(\cdot)}$ time, the gradient descent requires $n \cdot \mathtt{T}_{\nabla f_i(\cdot)}$ iteration complexity to compute the full gradient. Assuming that the problem is convex, L-smooth and μ-strongly convex, then this implies that the total complexity of gradient descent:

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

to get to an ε-solution is:

$$O\left(n \cdot \mathtt{T}_{\nabla f_i(\cdot)} \cdot \log \tfrac{1}{\varepsilon}\right).$$

(We neglect the presence of κ in the above expression for clarity). *But do we really need to compute the full gradient per iteration?*

Prototypical stochastic gradient descent (SGD). The natural question to ask is "*can we remove the n in the above complexity?*". This stems from the fact that we often have massive amounts of data: imagine that you are working with the ImageNet dataset, that has more than 14 million images. Having $n = 14 \cdot 10^6$ in the complexity above creates a huge burden in terms of computational resources.

What if we just computed the gradient on only one sample per iteration? I.e., we perform:

$$x_{t+1} = x_t - \eta \nabla f_{i_t}(x_t),$$

where $i_t \in [n]$ and is selected randomly. We can easily observe that, in this case, each iteration is very cheap, involving only the computation of the gradient $\nabla f_{i_t}(x_t)$, corresponding to one sample. Thus, someone would expect to obtain something like $O\left(\mathtt{T}_{\nabla f_i(\cdot)} \cdot \log \frac{1}{\varepsilon}\right)$, *but unfortunately this is not the case.* Discussing such trade-offs is the goal of this chapter: how we configure SGD algorithms is an active research area.

Some motivation for using SGD. There are both practical and theoretical motivations in using SGD in practice.

For practical motivation, we can easily construct cases that intuitively does not make sense using full gradient over SGD. E.g., consider the case where our dataset $\mathcal{D}$ has n copies of a single data sample w_1, y_1. In that case, $\nabla f(x) = \frac{1}{n} \cdot \sum_{i=1}^n \nabla f_i(x) = \frac{1}{n} \cdot n \cdot \nabla f_1(x) = \nabla f_1(x)$. Thus, $x_{t+1} = x_t - \eta \nabla f(x_t) = x_t - \eta \nabla f_{i_t}(x_t)$, but we have wasted $(n-1) \cdot \mathtt{T}_{\nabla f_i(\cdot)}$ complexity to compute the full gradient. This reasoning stems from the fact that, in many applications, it is not the case that all data points provide *new information*. For p dimensional least-squares objective, the complexity of computing the full gradient is $O(np)$, whereas $n \gg p$ is now often the case (big-data regime), computing the exact gradient might have diminishing returns. There might be cases where clusters of data can be well-represented by a single data point, and thus computing gradients for each data sample is a waste of computational resources. This issue is handled naturally with SGD.

The above can be empirically observed with real data. In the figure that follows from the writeup by Bottou, Curtis and Nocedal, it is obvious that SGD motions can be even faster than quasi-Newton methods, that exploit the curvature of the objective, beyond just the gradient information. (*Caveat: to*

[11] Usually, datasets are represented as $\{x_i, y_i\}_{i=1}^n$ in the ML literature, where x_i indicates the input and y_i the observed output. However, since we use x to denote the optimization variables in these notes, we use the less frequent notation of $\{w_i, y_i\}_{i=1}^n$ to denote the above.

achieve this plot though, heavy hyper-parameter tuning was required for SGD.)

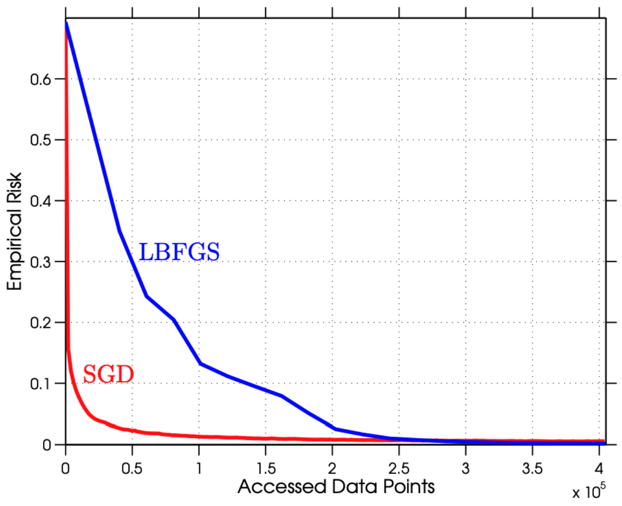

Fig. 45. Binary classification problem using logistic regression and the RCV1 dataset. Borrowed from Bottou, Curtis and Nocedal manuscript on "Optimization methods for large-scale machine learning" [50].

The theoretical motivation that stems from using SGD lies in that, while we often do not achieve in theory the same convergence rate than full gradient descent, SGD removes the n factor in the total complexity, creating an interesting trade-off for the practitioner. Moreover, it turns out that in the stochastic optimization setting, SGD yields the same convergence rate, even if we work in the *expected risk case.* Having such strong guarantees in the full gradient descent is not possible, as the latter is not even viable without the ability to compute the full gradient on, let's say, countably infinite data points that can be generated from the data distribution.

Some examples for SGD for common functions. We will consider the classical examples of linear and logistic regression.

Linear regression: In this well-studied setting, we can easily decompose the full gradient objective $\frac{1}{2}\|y - Ax\|_2^2$ into a summation of smaller objectives:

$$f(x) := \tfrac{1}{2}\|y - Ax\|_2^2 = \sum_{i=1}^{n} \tfrac{1}{2}\left(y_i - \alpha_i^\top x\right)^2 := \sum_{i=1}^{n} f_i(x).$$

Then, SGD looks like the following:

$$x_{t+1} = x_t - \eta\nabla f_{i_t}(x_t) = x_t + \eta\alpha_{i_t}(y_{i_t} - \alpha_{i_t}^\top x_t)$$

Logistic regression: In logistic regression, the objective function is similarly the summation of objectives obtained from samples:

$$f(x) := \tfrac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i\alpha_i^\top x)) := \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

As we shown in Chapter 2:

$$\nabla f_{i_t}(x_t) = \frac{-y_i}{1 + \exp(y_{i_t}\alpha_{i_t}^\top x_t)}\alpha_{i_t}$$

Then, SGD looks like the following:

$$x_{t+1} = x_t - \eta\nabla f_{i_t}(x_t) = x_t + \eta\frac{y_i}{1 + \exp(y_{i_t}\alpha_{i_t}^\top x_t)}\alpha_{i_t}$$

Neural network (MLP): Similar with logistic regression, but the objective function becomes the loss function of the neural network. Denote the weights and biases in a simple multi-layer neural-network as x, then sample prediction can be denoted as $\hat{y_i} = g_i(x)$. Take ℓ_2-norm loss as an example objective:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n}\|y_i - \hat{y_i}\|_2^2 = \frac{1}{n}\sum_{i=1}^{n}\|y_i - g_i(x)\|_2^2 = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

Then, SGD looks like the following:

$$x_{t+1} = x_t - \eta\nabla f_{i_t}(x_t; w_{i_t}, y_{i_t}),$$

where (w_{i_t}, y_{i_t}) is from the training set, and ∇f_{i_t} is usually referred to as the stochastic gradient.

Cut-off complexity per iteration for SGD. Figure 46 is from the process of Gradient Descent and Stochastic Gradient Descent on the contour of the parameter space, and the rows in the purple (GD) and blue columns (SGD) are data samples. In each step, GD computes the full gradient, and chooses the "most decreasing direction", perpendicular to the contour. SGD computes only part of the gradient by sub-selecting data samples, and takes a more winding path. Per iteration, the SGD minimizes a different objective function; that coming from randomly picked samples. This is not the same as optimizing the original objective, which is from the summation of samples, therefore SGD goes in random directions.

It is going to be proven in the next section that, overall, SGD moves in the correct direction *on expectation.* This is a nice feature that allows us to trade-off between high per-iteration complexity with exactness (full GD) and less per-iteration complexity with possibly more iterations. In practice, SGD is often used with momentum and weight-decay, in order to make use of the information from previous iterations. Also, the randomness of data selection can be implemented by randomly shuffling the dataset, and taking the data samples in sequence at each iteration. There are more sophisticated approaches of selecting samples, for example selecting the "harder" (not well predicted) samples from previous iterations, which requires more computation; or selecting data under fairness considerations.

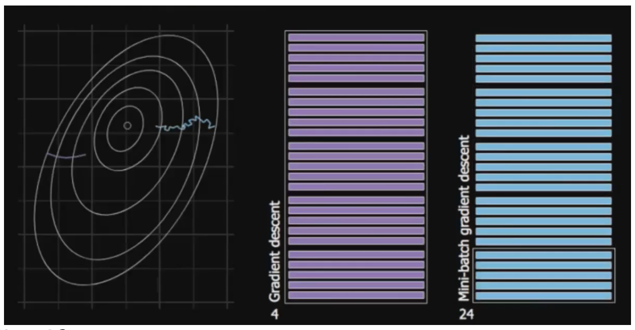

Fig. 46. SGD vs. GD on the parameter space contour (tribute to Piotr Skalski)

Basics of theory on SGD. Let us first define the notion of *unbiasedness*:

Definition 30. (Unbiasedness) *In statistics, we call the variable $\hat{\theta}$ an unbiased estimation of θ if:*

$$\mathbb{E}[\hat{\theta}] = \theta.$$

We will use the above definition to show that stochastic gradients are unbiased estimators of the full gradient. In particular, consider the case where $\nabla f(x) = \frac{1}{n}\sum_{i=1}^n f_i(x)$, similar to the empirical risk minimization case. Then:

Claim 7. *Let i be selected uniformly at random from the set $[n]$. Then, the atomic gradient $\nabla f_i(x)$ is an unbiased estimator of the true gradient, $\nabla f(x)$.*

Proof:

$$\begin{aligned}\mathbb{E}_i\left[\nabla f_i(x)\right] &= \sum_{j=1}^n \mathbb{P}\left[j=i\right]\nabla f_j(x)\\ &= \frac{1}{n}\sum_{j=1}^n \nabla f_j(x) = \nabla f(x).\end{aligned}$$

□

In this section, we would like to characterize theoretically the performance of SGD:

$$x_{t+1} = x_t - \eta_{i_t}\nabla f_{i_t}(x_t).$$

In order to limit the harmful effect of stochasticity, it is required to assume that the variance of $\nabla f_{i_t}(x_t)$ (its norm) is being bounded, or in other words one of the following inequalities is being assumed:

- There exist some $M, M_f \geq 0$ (*we often have small M close to 0 and M_f close to 1; and the constant M takes effect when the full gradient is 0, in which case the stochastic gradient is usually not 0 and still needs to be bounded*) such that:

$$\mathbb{E}_{i_t}\left[\|\nabla f_{i_t}(x_t)\|_2^2\right] \leq M + M_f\|\nabla f(x_t)\|_2^2;$$

or

- There exists a $C \geq 0$ such that:

$$\mathbb{E}_{i_t}\left[\|\nabla f_{i_t}(x_t)\|_2^2\right] \leq C.$$

The former means that norm of each individual gradient is not much greater than norm of the full gradient, while the latter implies that norm of each individual gradient is bounded with a constant.

SGD for smooth and strongly convex f with constant step size. We will focus on the following claim:

Claim 8. *Assume that f is a i) differentiable, ii) L-smooth, and iii) a μ-strongly convex function. There exist a constant $\eta \geq 0$ such that applying SGD from point x_0 with step size η, then:*

$$\mathbb{E}\left[f(x_{t+1}) - f(x^\star)\right] \leq (1-\mu\eta)^t\left(f(x_0) - f(x^\star)\right) + \mathcal{O}(\eta)$$

where $x^\star$ is the global minimizer of f.

Proof: By using the assumption of Lipschitz gradients, we have:

$$\begin{aligned}f(x_{t+1}) &\leq f(x_t) + \langle\nabla f(x_t), x_{t+1} - x_t\rangle + \tfrac{L}{2}\|x_{t+1} - x_t\|_2^2\\ &= f(x_t) + \langle\nabla f(x_t), x_t - \eta_t\nabla f_{i_t}(x_t) - x_t\rangle\\ &\quad + \tfrac{L}{2}\|x_t - \eta_t\nabla f_{i_t}(x_t) - x_t\|_2^2\\ &= f(x_t) - \eta\langle\nabla f(x_t), \nabla f_{i_t}(x_t)\rangle + \tfrac{L}{2}\eta^2\|\nabla f_{i_t}(x_t)\|_2^2\end{aligned}$$

Let's assume that indices $i_0, \ldots, i_{t-1}$ have been selected uniformly at random. So, taking the expectation with respect to i_t given $i_0, \ldots, i_{t-1}$ (for brevity we write $\mathbb{E}_{i_t}[.]$ instead of $\mathbb{E}_{i_t|i_{t-1},\ldots,i_0}[.]$):

$$\begin{aligned}\mathbb{E}_{i_t}\left[f(x_{t+1})\right] &\leq f(x_t) - \eta\left\langle\nabla f(x_t), \mathbb{E}_{i_t}\left[\nabla f_{i_t}(x_t)\right]\right\rangle\\ &\quad + \tfrac{L}{2}\eta^2\mathbb{E}_{i_t}\left[\|\nabla f_{i_t}(x_t)\|_2^2\right]\\ &\leq f(x_t) - \eta\|\nabla f(x_t)\|_2^2\\ &\quad + \tfrac{L}{2}\eta^2(M + M_f\|\nabla f(x_t)\|_2^2)\end{aligned}$$

which is the result of unbiasedness and boundedness of each individual gradient norm. Hence:

$$\begin{aligned}\mathbb{E}_{i_t}\left[f(x_{t+1}) - f(x^\star)\right] &\leq (f(x_t) - f(x^\star)) + \tfrac{L\eta^2M}{2}\\ &\quad - (\eta - \tfrac{L\eta^2M_f}{2})\|\nabla f(x_t)\|_2^2 \qquad (\star)\end{aligned}$$

By strong convexity, we have:

$$\begin{aligned}&f(x_t) \leq f(x^\star) + \langle\nabla f(x^\star), x_t - x^\star\rangle + \tfrac{1}{2\mu}\|\nabla f(x_t) - \nabla f(x^\star)\|_2^2\\ &\quad\Rightarrow f(x_t) - f(x^\star) \leq \tfrac{1}{2\mu}\|\nabla f(x_t)\|_2^2\end{aligned}$$

Furthermore, for $\eta \leq \frac{2}{LM_f}$, it is guaranteed that $\eta - \frac{L\eta^2M_f}{2} \geq 0$. Combining the two above facts in $(\star)$, we have:

$$\begin{aligned}\mathbb{E}_{i_t}\left[f(x_{t+1}) - f(x^\star)\right] &\leq (f(x_t) - f(x^\star)) + \tfrac{L\eta^2M}{2}\\ &\quad - 2\mu\left(\eta - \tfrac{L\eta^2M_f}{2}\right)(f(x_t) - f(x^\star))\\ &= \left(1 - 2\mu\left(\eta - \tfrac{L\eta^2M_f}{2}\right)\right)(f(x_t) - f(x^\star))\\ &\quad + \tfrac{L\eta^2M}{2}\\ (\text{Assume } \eta = \tfrac{1}{LM_f}:) &= (1-\mu\eta)(f(x_t) - f(x^\star)) + \tfrac{L\eta^2M}{2}\end{aligned}$$

Repeating the chain for $i_{t-1}, \ldots, i_0$, we have:

$$\begin{aligned}\mathbb{E}\left[f(x_{t+1}) - f(x^\star)\right] &\leq (1-\mu\eta)^t(f(x_0) - f(x^\star))\\ &\quad + \sum_{j=0}^t(1-\mu\eta)^j\tfrac{L\eta^2M}{2}\\ &= (1-\mu\eta)^t(f(x_0) - f(x^\star))\\ &\quad + \tfrac{L\eta^2M}{2}.\tfrac{1-(1-\mu\eta)^{t+1}}{1-1+\mu\eta}\\ (\text{Assume } \eta \leq \frac{1}{\mu}:) &= (1-\mu\eta)^t(f(x_0) - f(x^\star))\\ &\quad + \tfrac{L\eta M}{2\mu}(= \mathcal{O}(\eta))\end{aligned}$$

Note that the above result holds for $\eta \leq \min\left\{\frac{1}{LM_f}, \frac{1}{\mu}\right\}$. □

According to the above proof, we observe:

1. Fast linear convergence is connected with smaller $(1-\mu\eta)$ which is subject to selecting (valid) larger value of η.
2. After large enough iterations, the algorithm converges around a neighborhood of radius $\mathcal{O}(\eta)$.
3. When we do full gradient descent, $M = 0, M_f = 1$.
4. Smaller step sizes (η) yield better convergence, although slower, so there is a trade-off between accuracy and speed of the algorithm.

SGD for smooth and strongly convex f with decreasing step sizes. Here, we assume the simpler case:

$$\mathbb{E}_{i_t}\left[\|\nabla f_{i_t}(x_t)\|_2^2\right] \leq G^2$$

Claim 9. *Assume that f is a i) differentiable, ii) L-smooth, and iii) μ-strongly convex function. If we apply SGD starting from point x_1 with step size $\eta_t = \frac{1}{\mu t}$, then:*

$$\mathbb{E}\left[\|x_{t+1} - x^\star\|_2^2\right] \leq \frac{\max\{\|x_1 - x^\star\|_2^2, \frac{G^2}{\mu^2}\}}{t+1} =: \frac{\Delta}{t+1}$$

where $x^\star$ is the global minimizer of f.

Proof: We know that:

$$\mathbb{E}_{i_t}\left[\|x_{t+1} - x^\star\|_2^2\right] \leq \|x_t - x^\star\|_2^2 + \eta_t^2 \mathbb{E}_{i_t}\left[\|\nabla f_{i_t}(x_t)\|_2^2\right] - 2\eta_t \left\langle \mathbb{E}_{i_t}\left[\nabla f_{i_t}(x_t)\right], x_t - x^\star\right\rangle$$

By strong convexity:

$$\langle \nabla f(x) - \nabla f(x^\star), x - x^\star\rangle = \langle \nabla f(x), x - x^\star\rangle \geq \mu \|x - x^\star\|_2^2$$

Then:

$$\begin{aligned}\mathbb{E}_{i_t}\left[\|x_{t+1} - x^\star\|_2^2\right] &\leq \|x_t - x^\star\|_2^2 + \eta_t^2 G^2 - 2\eta_t \mu \|x_t - x^\star\|_2^2 \\ &= (1 - 2\eta_t\mu)\|x_t - x^\star\|_2^2 + \eta_t^2 G^2 \quad (\star\star)\end{aligned}$$

Finally, in order to prove the claim 9, let's use induction. First, it is trivial that:

$$\mathbb{E}\left[\|x_1 - x^\star\|_2^2\right] = \|x_1 - x^\star\|_2^2 \leq \frac{\max\{\|x_1 - x^\star\|_2^2, \frac{G^2}{\mu^2}\}}{1}$$

Assume that for t:

$$\mathbb{E}\left[\|x_t - x^\star\|_2^2\right] \leq \frac{\max\{\|x_1 - x^\star\|_2^2, \frac{G^2}{\mu^2}\}}{t} = \frac{\Delta}{t}$$

Using the chain of expectations, according to assumption of the induction and $(\star\star)$, we have:

$$\begin{aligned}\mathbb{E}\left[\|x_{t+1} - x^\star\|_2^2\right] &\leq \left(1 - \frac{2}{t}\right)\mathbb{E}\left[\|x_t - x^\star\|_2^2\right] + \frac{G^2}{\mu^2 t^2} \\ &\leq \left(1 - \frac{2}{t}\right)\frac{\Delta}{t} + \frac{G^2}{\mu^2 t^2} \\ &\leq \left(\frac{1}{t} - \frac{2}{t^2}\right)\Delta + \frac{\Delta}{t^2} \\ &= \left(\frac{1}{t} - \frac{1}{t^2}\right)\Delta \\ &\leq \frac{\Delta}{t+1}\end{aligned}$$

□

Comparing GD and SGD. We studied both gradient descent and its stochastic version, carefully for the specific case of strongly convex functions. Similar arguments (with corresponding changes in the convergence rates) can be made also for the cases of just convex L-smooth functions, as well as for general smooth non-convex functions; we skip all these cases, since –for the purpose of teaching– do not add much to the discussion we have here.

As we showed in claim 8, for the ideal case of smooth and strong convex function, having the linear convergence rate, there is a trade-off between the accuracy and the speed. Then in claim 9, using a decreasing learning rate, we guaranteed accuracy of SGD, but lost the linear convergence rate. While, in previous chapters, we showed that GD, in addition to keeping the accuracy, exploits a linear convergence rate. So, the number of samples plays a critical role in making the decision whether to choose GD or SGD: In other words, although GD converges with less number of iterations to global minimizer of the function, each iteration can consume a considerable portion of the time. SGD, needs more iterations to converge to the global minimum, but it needs less complexity per iteration. Thus, here, the theory justifies what we intuitively expect from stochastic and full gradient descent.

	iteration complexity	per-iteration cost	total cost
batch GD	$\log\frac{1}{\varepsilon}$	n	$n\log\frac{1}{\varepsilon}$
SGD	$\frac{1}{\varepsilon}$	1	$\frac{1}{\varepsilon}$

The above table shows the computational complexity of batch GD and GD. The overall cost in the right column is computed by multiplying iteration complexity and per-iteration cost. Therefore the real comparison is between $n\log\frac{1}{\varepsilon}$ and $\frac{1}{\varepsilon}$. In a practical sense the algorithm does not have to be "too accurate" (see e.g., the case of neural networks), for example $\epsilon = 10^{-3}$ and $\frac{1}{\epsilon} = 1000$, meanwhile n is often the order of millions. This makes SGD more preferable compared to batch GD. Another reason that people prefer to use SGD is that it has a better generalization capability (out of scope for this class).

Variants of SGD. *Mini-batch SGD* Note that instead of these two methods, we can use a mini-batch of samples to do a stochastic gradient descent in each iteration, but using the expectation analysis regime, the rate of convergence does not change.

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

In practice, we rarely use single data point SGD (batch size = 1) because the update direction is arbituary and the variance compared to the true gradient is too high.

SGD with importance sampling. As mentioned in a previous section, to perform SGD with importance sampling, we have to "carefully" select the next sample, which requires more computing each iteration. Hence there is a trade-off between per-iteration complexity and possible less iterations.

Stochastic variance-reduced gradient (SVRG). The intuition is to construct a hybrid version of SGD and GD.

$$x_{t+1} = x_t - \eta_t\left(\nabla f_{i_t}(x_t) - \left(\nabla f_{i_t}(\widetilde{x}_q) - \nabla f(\widetilde{x}_q)\right)\right)$$

In addition to SGD, a bias correction term for gradient estimate is computed every few iterations for SVRG, which means the full-gradient has to be computed by some interval. There is also theoretical guarantees that SVRG can achive same convergence rate as GD with no additive error.

Coordinate descent. Similar to what we did for samples, the idea of coordinate descent is to update just one feature (or a subset of features) in each iteration. This method seems interesting specially for the cases with $p \gg 1$ (high-dimensional case, not enough data). *What if we just computed the gradient on only one feature per iteration?* I.e., we perform:

$$(x_{t+1})_{i_t} = (x_t)_{i_t} - \eta_t \nabla_{i_t} f(x_t),$$

where $i_t \in [p]$ and is selected randomly. Regarding the fact that in each iteration, we just compute the gradient with respect to one coordinate, $\nabla_{i_t} f(x_t)$, each iteration is cheaper than its counterpart in GD. (Something like $O\left(\mathtt{T}_{\nabla_i f(\cdot)} \cdot \log \frac{1}{\varepsilon}\right)$, is expected.)

Appendix

1. J. Nocedal and S. Wright. Numerical optimization. Springer Science & Business Media, 2006.
2. Y. Nesterov. Introductory lectures on convex optimization: A basic course, volume 87. Springer Science & Business Media, 2013.
3. S. Boyd and L. Vandenberghe. Convex optimization. Cambridge university press, 2004.
4. D. Bertsekas. Convex optimization algorithms. Athena Scientific Belmont, 2015.
5. Sébastien Bubeck. Convex optimization: Algorithms and complexity. Foundations and Trends® in Machine Learning, 8(3-4):231–357, 2015.
6. S. Weisberg. Applied linear regression, volume 528. John Wiley & Sons, 2005.
7. T. Hastie, R. Tibshirani, and M. Wainwright. Statistical learning with sparsity: the lasso and generalizations. CRC press, 2015.
8. J. Friedman, T. Hastie, and R. Tibshirani. The elements of statistical learning, volume 1. Springer series in statistics New York, 2001.
9. M. Paris and J. Rehacek. Quantum state estimation, volume 649. Springer Science & Business Media, 2004.
10. M. Daskin. A maximum expected covering location model: formulation, properties and heuristic solution. Transportation science, 17(1):48–70, 1983.
11. I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. MIT press, 2016.
12. L. Trefethen and D. Bau III. Numerical linear algebra, volume 50. Siam, 1997.
13. G. Strang. Introduction to linear algebra, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.
14. G. Golub. Cmatrix computations. The Johns Hopkins, 1996.
15. A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
16. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
17. S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, 2015.
18. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013.
19. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.
20. Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1243–1252. JMLR. org, 2017.
21. Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In Fifteenth annual conference of the international speech communication association, 2014.
22. Tom Sercu, Christian Puhrsch, Brian Kingsbury, and Yann LeCun. Very deep multilingual convolutional neural networks for LVCSR. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4955–4959. IEEE, 2016.
23. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. page arXiv:1706.03762, 2017.
24. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. page arXiv:1810.04805, 2018.
25. Luowei Zhou, Hamid Palangi, Lei Zhang, Houdong Hu, Jason J Corso, and Jianfeng Gao. Unified vision-language pre-training for image captioning and VQA. In AAAI, pages 13041–13049, 2020.
26. Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.
27. Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. arXiv preprint arXiv:1909.08053, 2019.
28. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv preprint arXiv:1910.10683, 2019.
29. Gary Marcus, Ernest Davis, and Scott Aaronson. A very preliminary analysis of DALL-E 2. arXiv preprint arXiv:2204.13807, 2022.
30. John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. Nature, 596(7873):583–589, 2021.
31. Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.
32. Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. arXiv preprint arXiv:2004.08900, 2020.
33. H. Karimi, J. Nutini, and M. Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 795–811. Springer, 2016.
34. Philip Wolfe. Convergence conditions for ascent methods. SIAM review, 11(2):226–235, 1969.
35. Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. Pacific Journal of mathematics, 16(1):1–3, 1966.
36. Stephen Wright and Jorge Nocedal. Numerical optimization. Springer Science, 35(67-68):7, 1999.
37. B. Polyak. Introduction to optimization. Inc., Publications Division, New York, 1, 1987.
38. Stephen Boyd, Lin Xiao, and Almir Mutapcic. Subgradient methods. lecture notes of EE392o, Stanford University, Autumn Quarter, 2004:2004–2005, 2003.
39. Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. Naval research logistics quarterly, 3(1-2):95–110, 1956.
40. M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Proceedings of the 30th international conference on machine learning, number CONF, pages 427–435, 2013.
41. J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In Proceedings of the 25th international conference on Machine learning, pages 272–279, 2008.
42. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, 2009.
43. A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In Advances in neural information processing systems, pages 1257–1264, 2008.
44. T. Booth and J. Gubernatis. Improved criticality convergence via a modified Monte Carlo power iteration method. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
45. S. Zavriev and F. Kostyuk. Heavy-ball method in nonconvex optimization problems. Computational Mathematics and Modeling, 4(4):336–341, 1993.
46. E. Ghadimi, H. Feyzmahdavian, and M. Johansson. Global convergence of the heavy-ball method for convex optimization. In 2015 European control conference (ECC), pages 310–315. IEEE, 2015.
47. Y. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. In Dokl. akad. nauk Sssr, volume 269, pages 543–547, 1983.
48. B. O'Donoghue and E. Candes. Adaptive restart for accelerated gradient schemes. Foundations of computational mathematics, 15(3):715–732, 2015.
49. O. Devolder, F. Glineur, and Y. Nesterov. First-order methods of smooth convex optimization with inexact oracle. Mathematical Programming, 146(1-2):37–75, 2014.
50. L. Bottou, F. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. Siam Review, 60(2):223–311, 2018.