IFT 6085 - Lecture 6
Nesterov's Momentum, Stochastic Gradient Descent

This version of the notes has not yet been thoroughly checked. Please report any bugs to the scribes or instructor.

Scribes:
Winter 2019: Yann Bouteiller & Remi Piche-Taillefer
Winter 2018: Charles Ashby & Gabriele Prato

Instructor: Ioannis Mitliagkas

1 Summary

This lecture covers the following elements of optimization theory:

- Failing case of Polyak's momentum
- Nesterov's momentum
- Stochastic gradient descent

Most of the lecture has been adapted from Bubeck [1], Lessard et al. [2], Nesterov [3] and Shalev-Shwartz S. [4].

2 Failing case of Polyak's Momentum

In the previous lecture we presented Polyak's momentum algorithm (or heavy-ball method), in which the iteration step is given by:

$$x_{t+1} = x_t - \gamma \nabla f(x_t) + \mu(x_t - x_{t-1}), \quad \mu \in [0,1], \gamma > 0 \tag{1}$$

This method is known to be robust and effective in many real-world applications. However, in a paper published in 2015, Lessard et al. [2] have shown that there exist strongly-convex and smooth functions for which, by choosing carefully the hyperparameters γ and μ, the heavy-ball method fails to converge. We give here merely the intuition behind the malfunction of the algorithm as well as empirical results taken from the paper.

Lessard et al. [2] give the following one-dimensional example :
Let f be a piece-wise quadratic function whose gradient is:

$$\nabla f(x) = \begin{cases} 25x & \text{if } x < 1 \\ x + 24 & \text{if } 1 \le x < 2 \\ 25x - 24 & \text{otherwise} \end{cases}$$

Since $\forall x_1, x_2, \|\nabla f(x_1) - \nabla f(x_2)\| \le 25\|x_1 - x_2\|$, we can infer that f is 25-smooth. Let us note the following quantities :

$$\nabla(f(x) - \frac{\|x\|^2}{2}) = \nabla f(x) - x = \begin{cases} 24x & \text{if } x < 1 \\ 24 & \text{if } 1 \le x < 2 \\ 24x - 24 & \text{otherwise} \end{cases}$$

$$\nabla^2(f(x) - \frac{\|x\|^2}{2}) = \begin{cases} 24 & \text{if } x < 1 \\ 0 & \text{if } 1 \le x < 2 \\ 24 & \text{otherwise} \end{cases}$$

$\nabla^2(f(x) - \frac{\|x\|^2}{2})$ is non-negative, so f is 1-strongly convex.

Below are displayed the first 50 iterates of Polyak's momentum algorithm applied to f, using $\mu = \frac{4}{9}$, $\gamma = \frac{1}{9}$ and $x_0 = 3.3$. The output values are in fact bound to cycle through these 3 points indefinitly.

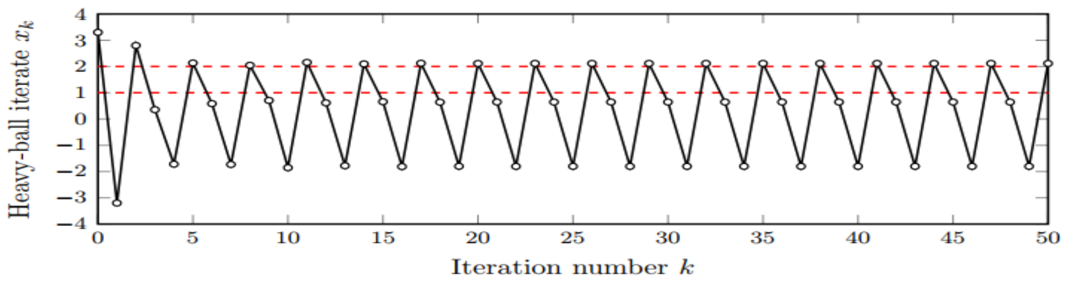

Figure 1: Value of x_k for the 50 first iterations of Polyak's momentum algorithm. The dashed red lines separate the pieces of $f(x)$. Taken from Lessard et al. [2].

Despite the fact that f is 1-strongly convex and 25-smooth, Polyak's momentum algorithm loops indefinitely over these points. Lessard et al. [2]'s proof revolves around the idea that the iterates are stuck in a limit cycle. The authors show that the following sub-series of the iterates tend to three points that are not the optimal value of f.

$$x_{3n} \to p, \quad x_{3n+1} \to q, \quad x_{3n+2} \to r$$

They show that $p \approx 0.65$, $q \approx -1.80$ and $r \approx 2.12$. Figure 2 plots $f(x)$ for these 3 points.

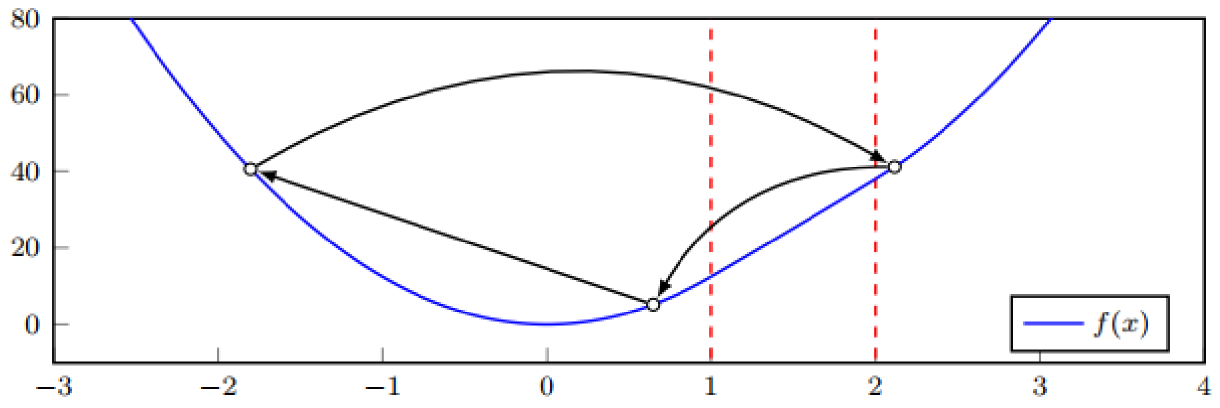

Figure 2: Illustration of the limit values of the failing case of Polyak's momentum algorithm, the dashed red lines separate the pieces of f. Image taken from Lessard et al. [2].

3 Nesterov's Accelerated Gradient Descent

3.1 Motivation

In the last section, we saw that Polyak's momentum algorithm can fail to converge for some carefully built convex optimization problems. Leveraging the idea of momentum introduced by Polyak, Nesterov introduced a slightly altered update rule that has been shown to converge not only for quadratic functions, but for general convex functions. Despite extending the hypothesis space for f, this algorithm still manages to achieve an acceleration rate comparable to the heavy-ball method. The main ideas of this demonstration are presented at the end of this section.

The Nesterov's Accelerated Gradient algorithm is described as follow by Sutskever et al. [5]:

Algorithm 1 Nesterov's Accelerated Gradient Descent

```
**Require:** training steps T, learning rate $\gamma$, momentum $\mu$ and parameter's initialization $x_0$.
$v_0 \leftarrow 0$
**for** $t \leftarrow 0$ **to** $T-1$ **do**
    $v_{t+1} = \mu v_t - \gamma \nabla f(x_t + \mu v_t)$
    $x_{t+1} = x_t + v_{t+1}$
**end**
**return** $x_T$
```

The v_{t+1} displacement vector is calculated by applying the μ momentum to the previous v_t displacement, and subtracting the gradient step to $x_t + \mu v_t$, which is the point where the momentum term leads from x_t. Rewriting these two sequences as one yields:

$$x_{t+1} = x_t + \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1})) \tag{2}$$

Comparing equation 1 with equation 2, we can see that Polyak's method evaluates the gradient before adding momentum, whereas Nesterov's algorithm evaluates it after applying momentum, which intuitively brings us closer to the minimum x^*, as illustrated by figure 3.

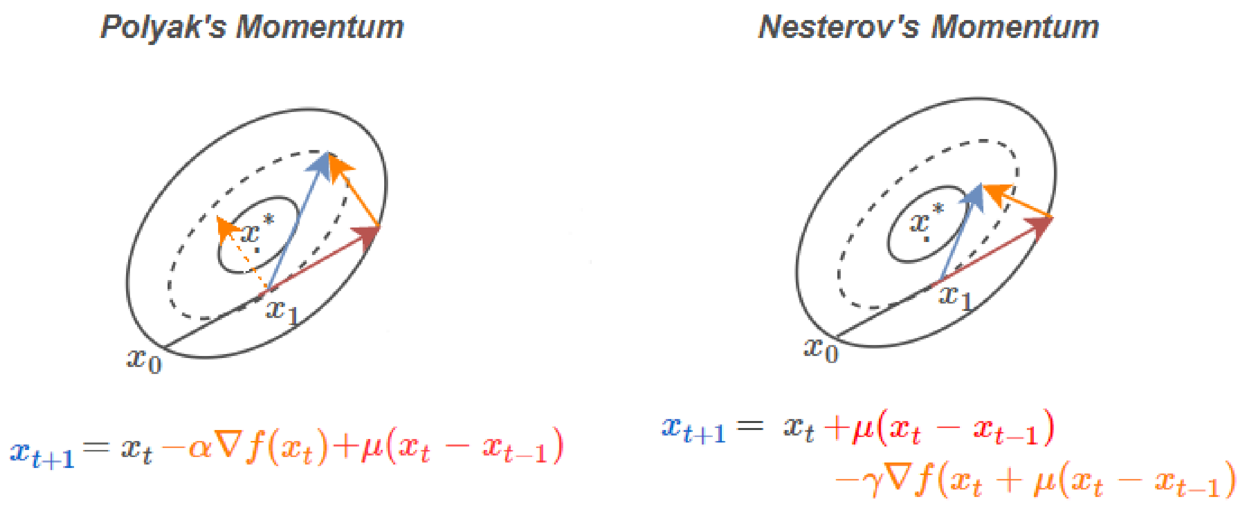
Polyak's Momentum
Nesterov's Momentum
x^*
x_1
x_0
x^*
x_1
x_0
$x_{t+1} = x_t - \alpha \nabla f(x_t) + \mu(x_t - x_{t-1})$
$x_{t+1} = x_t + \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1})$

Figure 3: Comparison between Polyak's and Nesterov's momentum. The gradient descent step (orange arrow) is perpendicular to the level set before applying momentum to x_1 (red arrow) in Polyak's algorithm, whereas it is perpendicular to the level set after applying momentum to x_1 in Nesterov's algorithm.

The convergence rate (upper bound on the sub-optimality) for different classes of functions for gradient descent and Nesterov's accelerated gradient descent are compared below.

Table 1: Convergence rate for Gradient Descent & Nesterov's Accelerated Gradient

Class of Function	GD	NAG
Smooth	$O(1/T)$	$O(1/T^2)$
Smooth & Strongly-Convex	$O\left(exp\left(-\frac{T}{\kappa}\right)\right)$	$O\left(exp\left(-\frac{T}{\sqrt{\kappa}}\right)\right)$

Nesterov's Augmented Gradient in the case of smooth and strongly convex functions gives the acceleration that we had with Polyak's momentum for quadratic functions. This is great, because we get the guarantee for a more general class of functions. Recall that if f is α-strongly convex and β-smooth then $\kappa = \frac{\beta}{\alpha}$. Therefore, when $\beta << \alpha$ (i.e. some dimensions have very steep gradients), the acceleration becomes very significant.

3.2 Main Ideas of the Convergence Proof for Nesterov's Accelerated Gradient Descent

In this section, we present the main items behind the proof of convergence for Nesterov's momentum algorithm for general convex functions. In order to achieve this proof, Nesterov uses an **estimate sequence**.

Definition 1 (Estimate Sequence - Definition 2.2.1 from Nesterov's book [3]). *A pair of sequences* $\{\phi_k(x)\}_{k=0}^{\infty}$ *and* $\{\lambda_k\}_{k=0}^{\infty}$ *where* $\lambda_k \geq 0$*, is called an Estimate Sequence of* f *if the following conditions hold:*

- $\lambda_k \to 0$ *when* $k \to \infty$
- $\forall x \in \mathbb{R}^n, \forall k \geq 0$*, the following holds:*

$$\phi_k(x) \leq (1 - \lambda_k) f(x) + \lambda_k \phi_0(x)$$

Figure 4 illustrates the way an estimate sequence aligns with the f function. From definition 1 we can observe that:

$$\forall x \in \mathbb{R}^n, \lim_{k \to \infty} \phi_k(x) \leq f(x) \tag{3}$$

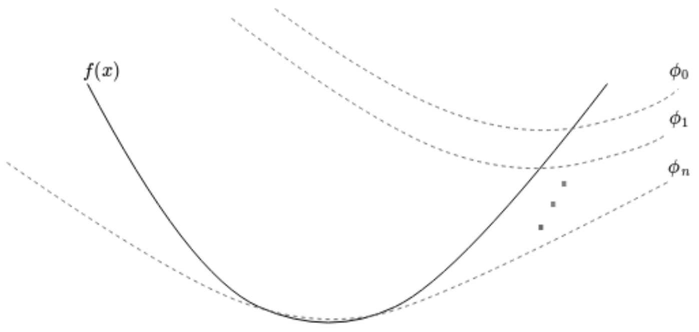

Figure 4: Example of an estimate sequence

When $k \to \infty$, ϕ becomes a lower bound of f, since $\lambda_k \phi_0(x) \to 0$ and $(1 - \lambda_k) f(x) \to f(x)$.
The idea is to find an estimate sequence ϕ to help us minimize f. To do so, the author introduces the following result:

Theorem 2 (Lemma 2.2.1 of Nesterov's book [3]). *Assuming that the estimate sequence is such that* $\forall k$, $f(x_k) \leq \phi_k^*$ *where* $\phi_k^* = \min_x \phi_k(x)$*, then:*

$$f(x_k) - f(x^*) \leq \lambda_k \left[\phi_0(x^*) - f(x^*)\right]$$

Since $\phi_0(x^*) - f(x^*)$ is a fixed quantity, the rate of convergence for the sub-optimality will depend on how quickly λ_k decreases to 0. Figure 5 presents a visualization of lemma 2.2.1.

Let us first define our first elements in the sequence.

$$\phi_0(x) = f(x_0) + \frac{\alpha}{2}\|x - x_0\|^2$$

$$\lambda_0 = 1$$

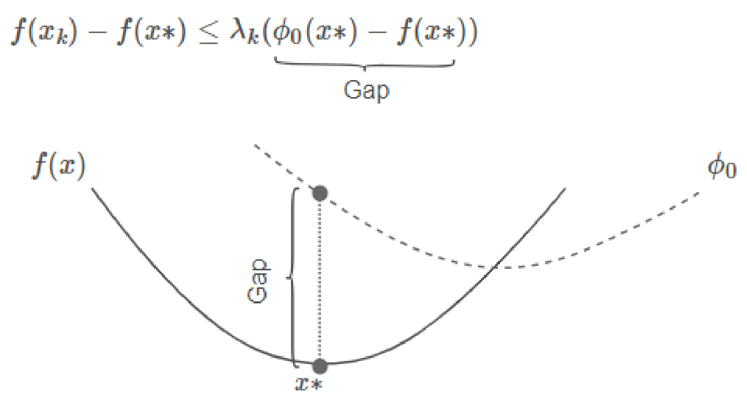
$f(x_k) - f(x*) \leq \lambda_k \underbrace{(\phi_0(x*) - f(x*))}_{\text{Gap}}$
$f(x)$
ϕ_0
Gap
$x*$

Figure 5: Illustration of lemma 2.2.1

The next elements are of the following form:

$$\phi_{k+1}(x) = (1-\alpha_k)\phi_k(x) + \alpha_k \left(f(y_k) + \langle \nabla f(y_k), x - y_k \rangle + \frac{\alpha}{2}\|x - y_k\|^2 \right)$$

$$\lambda_{k+1} = (1-\alpha_k)\lambda_k$$

Where $\{y_k\}_{k=0}^{\infty}$ is an arbitrary sequence, and $\{\alpha_k\}_{k=0}^{\infty}$ is a sequence such that $\alpha_k \in (0,1)$ and $\sum_{k=0}^{\infty} \alpha_k = \infty$. This carefully designed sequence is then used to show how λ_k can converge efficiently, and how the minimum point of ϕ_k converges to x^*.

4 Stochastic Gradient Descent

4.1 Motivation

As we have seen in the previous lecture, Gradient Descent is not subject to variance since at every step we compute the average gradient using the whole dataset. The downside is that every step is very computationally expensive, $O(nd)$ per iteration, where n is the number of samples in our dataset and d is the number of dimensions of x. Since we need $O(d)$ iterations to converge, the total problem cost is about $O(nd^2)$.

Gradient Descent becomes impractical when dealing with large datasets. This is where Stochastic Gradient Descent comes in. It is a modified version of Gradient Descent which does not use the whole set of examples to compute the gradient at every step. By doing so, we can reduce computation all the way down to $O(d)$ per iteration, instead of $O(nd)$.

Algorithm 2 Stochastic Gradient Descent

```
Require: training time T, learning rate $\gamma$, batch size $K$ and parameter's initialization $x_0$.
for $t \leftarrow 0$ to $T-1$ do
    Sample $(i_1, ..., i_K) \sim U^K(1, ..., n)$
    $x_{t+1} = x_t - \gamma \frac{1}{K} \sum_{j=1}^{K} \nabla f_{i_j}(x_t)$
end
return $x_{T-1}$
```

Note that in expectation, we converge like gradient descent, since $\mathbb{E}_{i \sim U(1,...,n)}[\nabla f_i(x_t)] = \nabla f(x_t)$, therefore, the expected iterate of SGD converges to the optimum. The downside is that stochasticity brings variance. Even if SGD is shown to converge, the variance can seriously handicap the convergence rate as we will see. This is especially true the smaller the batch size is, as variance is inversely proportional to the number of examples used to compute the gradient at every step.

SGD's convergence rate for Lipschitz & convex functions is $O(\frac{1}{\sqrt{T}})$ and $O(\frac{1}{T})$ for strongly convex. More iterations are needed to reach the same accuracy as GD, but the iterations are far cheaper.

4.2 Bias-Variance Trade-off

Recall from the last lecture that the iteration step at time t for Gradient Descent is given by $x_{t+1} = x_t - \gamma \nabla f(x_t)$ where $\nabla f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$. The variance from SGD comes from the fact that we do not take the average gradient over the whole dataset.

When training begins, the model weights x are more than likely very far from the optimal value $x* = \min_x f(x)$, therefore, most of our data points x_i, agree on the direction of the gradient. (see the high bias region in Figure 6)

However, when training goes on, most of the model weights are already close to the optimum, some are well tuned and others are not, this is where the variance has a significant impact on our convergence rate because almost all samples in our batch will produce gradients pointing in different directions, therefore, if we keep the same step size we will bounce around the optimum. This is called the noise ball effect.

To avoid bouncing around the minimum, we can use a decaying step size. The blue line in Figure 6 illustrates what happens when we reduce the variance by decaying the step size, another trick to reduce the variance would be to increase the batch size (in the limit we would get the same convergence rate as Gradient Descent).

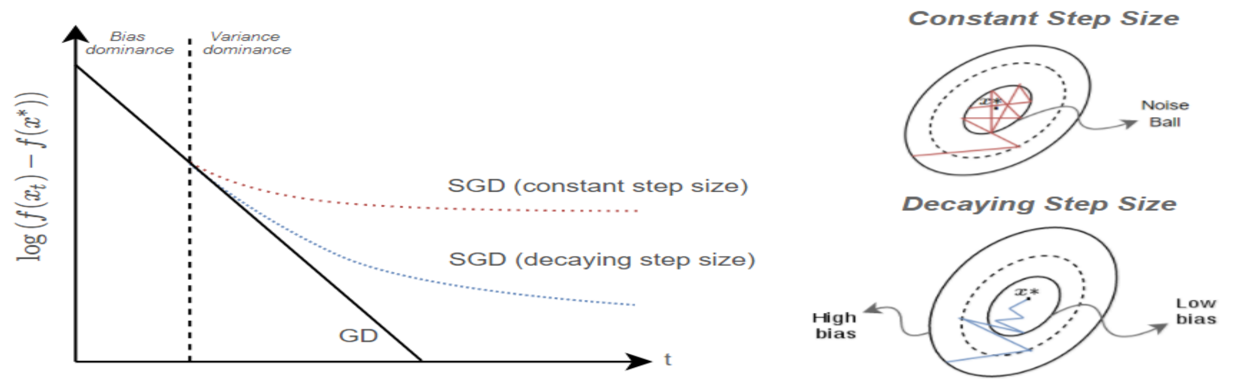

Figure 6: Convergence for Gradient Descent and Stochastic Gradient Descent. The blue line on the left plot illustrates what happens when we decay the step size. The red line shows what happens with constant step size, notice how it flattens out on the left plot.

References

[1] S. Bubeck. Convex Optimization: Algorithms and Complexity. *ArXiv e-prints*, Nov. 2015.

[2] L. Lessard, B. Recht, and A. Packard. Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints. *ArXiv e-prints*, Aug. 2014.

[3] Y. Nesterov. Introductory lecture on convex programming. *ISBN*, 1998.

[4] B.-D. S. Shalev-Shwartz S. Understanding machine learning: From theory to algorithms. *Cambridge*, 2014.

[5] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. *JMLR*, 2013.