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

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

Scribe(s): 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 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 took a look at Polyak's momentum algorithm (or heavy-ball method), with iteration step given by:

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

In their 2015 paper, Lessard et al. [2] show that there exists a convex function $f(x)$ and specific hyperparameters choices for which the heavy-ball method fails to converge. We give here only the intuition behind the malfunction of the algorithm as well as empirical results taken from the paper because the whole proof is out of the scope of this lecture.

Let f be defined by its gradient as:

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

Figure 1 displays the first 50 iterates of Polyak's momentum algorithm using $\mu = \frac{4}{9}$, $\alpha = \frac{1}{9}$ and $x_0 = 3.3$.

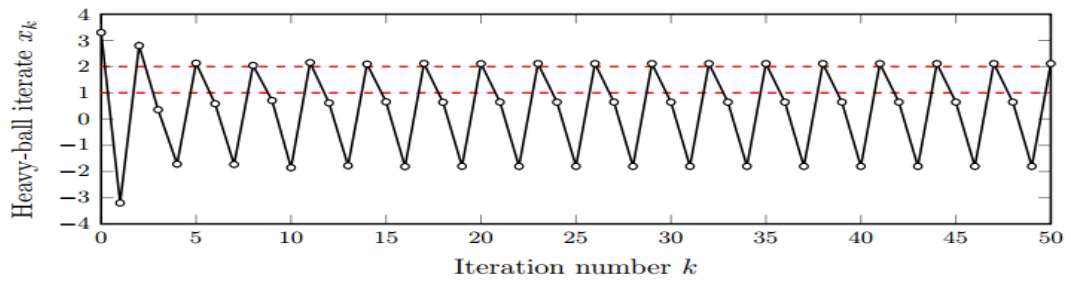

Figure 1: Value of x_k for the 50 first iterations of Polyak's momentum algorithm. We can see that the values bounce between 3 points, the dashed red lines separate the pieces of $f(x)$. Taken from Lessard et al. [2].

The proof revolves around the idea that the iterates are stuck in a limit cycle. To prove that, 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.

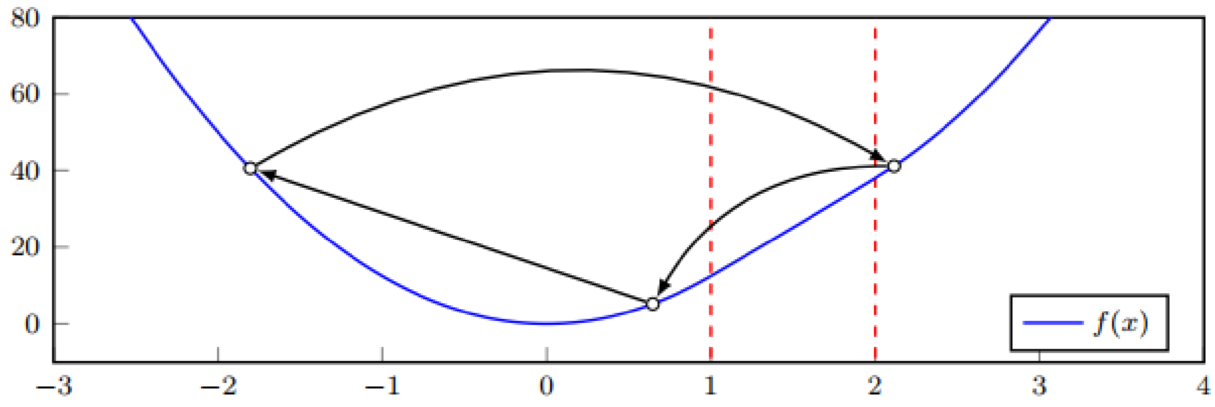

Figure 2: Illustration of the limit values of the failing case of Polyak's momentum algorithm, the dashed red lines separate the pieces of f. Image taken from Lessard et al. [2].

3 Nesterov Accelerated Gradient

3.1 Motivation

In the last section, we saw that Polyak's momentum algorithm can fail to converge for relatively simple convex optimization problems (it can be shown that the counter example we presented is a strongly-convex function). Leveraging the idea of momentum introduced by Polyak, Nesterov solved that problem by finding an algorithm achieving the same acceleration as the heavy-ball method, but that can be shown to converge for general convex functions. We show a glimpse of the proof of convergence at the end of this section.

Algorithm 1 Nesterov Accelerated Gradient

```
Require: training time T, learning rate $\gamma$, momentum $\mu$ and parameter's initialization $x_0$.
$y_0 \leftarrow x_0$
for $t \leftarrow 0$ to $T-1$ do
    $y_{t+1} = x_t - \gamma \nabla f(x_t)$
    $x_{t+1} = y_{t+1} + \mu(y_{t+1} - y_t)$
end
return $x_{T-1}$
```

The main difference between Polyak's and Nesterov's algorithm come from the fact that the latter has two sister sequences working together to produce a "corrective" movement like we can see in figure 3.

Rewriting these two sequences as one yields:

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

Comparing with (1), we see that Polyak's method evaluates the gradient before adding momentum, while Nesterov's algorithm evaluates the gradient after applying momentum.

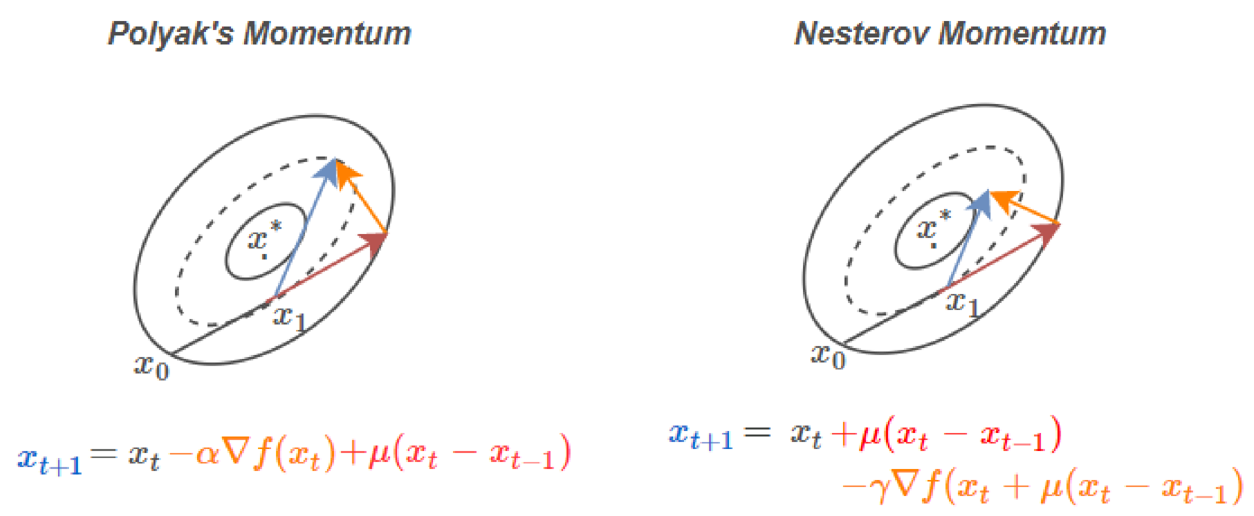
Polyak's Momentum
Nesterov Momentum
x^*
x_1
x_0
$x_{t+1} = x_t - \alpha \nabla f(x_t) + \mu(x_t - x_{t-1})$
$x_{t+1} = x_t + \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1})$

Figure 3: Comparison between Polyak and Nesterov momentum. Notice how the gradient step with Polyak's momentum is always perpendicular to the level set.

Table 1 gives the convergence rate (upper bound on the sub-optimality) for different classes of functions for gradient descent and Nesterov accelerated gradient.

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

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

As we can see, Nesterov Augmented Gradient gives acceleration in the case of smooth and strongly convex functions 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 curvature), the acceleration becomes very significant.

3.2 Convergence Proof for Nesterov Accelerated Gradient

In this section, we state the main theorems behind the proof of convergence for Nesterov Accelerated Gradient for general convex functions. The first tool we will need is called an **estimate sequence**.

Definition 1 (Estimate Sequence)**.** *A pair of sequences* $\{\phi_k(x)\}_{x=0}^{\infty}$ *and* $\{\lambda_k\}_{k=0}^{\infty}$ *where* $\lambda_k \geq 0$*, is called an Estimate Sequence if:*

- $\lambda_k \to 0$ *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 estimate sequences work, as we can see, if $\phi_0(x) \geq f(x)$ then when $k \to \infty$, $\phi(x) \to f$, $\forall x \in R^d$.

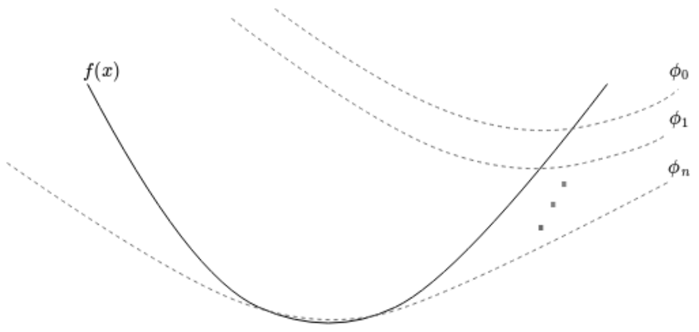

Figure 4: Illustration of an estimate sequence, we can see that when $k \to \infty$, ϕ gets closer and closer to f.

The idea is to find an estimate sequence ϕ to help us minimize f. To do so, the author introduce the following result:

Theorem 2 (Lemma 2.2.1 of Nesterov's Book [3]). *If we have $f(x_k) \leq \phi_k^*$ where $\phi_k^* = \min_{x \in R^n} \phi_k(x)$, 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, all that is left to do is find an estimate sequence ϕ_k and λ_k such that $f(x_k) \leq \phi_k^*$ to get an upper bound on the sub-optimality. Figure 5 presents a visualization of lemma 2.2.1.

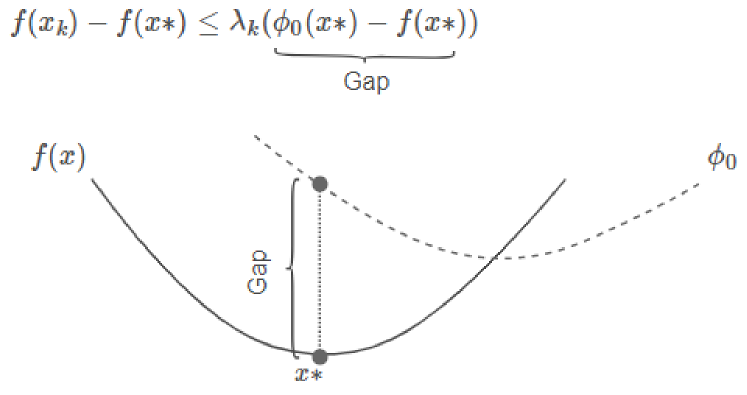
$f(x_k) - f(x*) \leq \lambda_k(\underbrace{\phi_0(x*) - f(x*)}_{\text{Gap}})$
$f(x)$
ϕ_0
Gap
$x*$

Figure 5: Illustration of lemma 2.2.1

The rest of the proof is all about putting these pieces together: designing an estimate sequence with the right rate, while guaranteeing that NAG satisfies the assumption of **Theorem 2** ($f(x_k) \leq \phi_k^*$).

4 Stochastic Gradient Descent

4.1 Motivation

Like we saw in the last 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's a modified version of Gradient Descent which doesn't 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 tunned and other's aren't, 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).

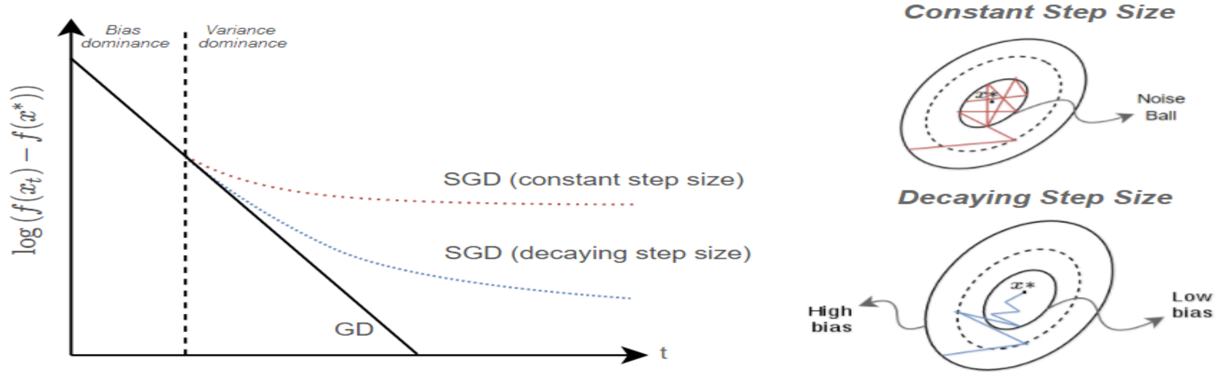

Figure 6: Convergence for Gradient Descent and Stochastic Gradient Descent. The blue line on the left plot illustrates what happens when we decay the step size. The red line shows what happens with constant step size, notice how it flattens out on the left plot.

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.