A Comparison Analysis of Accelegrad and UnixGrad Optimization Algorithms

Optimization

University of Padova - Master of Science in ICT for Internet and Multimedia

Francesco Mandruzzato - 1204532

francesco.mandruzzato.1@studenti.unipd.it

February 11, 2021

1 Accelegrad

1.1 Previous related work

The Accelerated Gradient Descent method devised by Nesterov is a popular optimization technique which provides faster convergence rate $\mathcal{O}(1/T^2)$ in the smooth case, guaranteeing sublinear convergence rate $\mathcal{O}(1/T)$ for a general convex objective function f with Lipschitz continuous gradient with constant $L > 0$. The structure of the algorithm is divided in two different steps:

1. **Extrapolation step**: We move along the direction of the difference between the last two iterates:
$$y_k = x_k + \beta_k(x_k - x_{k-1}), \tag{1}$$
were β_k is chosen depending on the properties of f. Ideally this parameter regulates the amount of information that we want to keep regarding past iterates.

2. **Gradient step** We performe a step likewise the GD algorithm in y_k to get the next point x_{k+1}:
$$x_{k+1} = y_k - \alpha_k \nabla f(y_k), \tag{2}$$
where $\alpha_k = 1/L$

Despite its efficiency, Accelerated Gradient Descent is inappropriate for handling noisy feedback because of the choice of the step size α_k and acceleration requires the knowledge of the objective's smoothness.

One of the main popular adaptive first order methods comes with AdaGrad [1]. Informally, AdaGrad employs the geometry of the dataset in the sense that the adaptation of the step size α_k vary according with the dimension considered, in other words, it associates a small step size

to frequently occurring features and a large step size to infrequent features. This adaptation is encoded in its update rule:

$$x_{t+1} = \Pi_K^{G_t^{1/2}}(x_t - \eta G_t^{-1/2} g_t), \tag{3}$$

where $G_t = \sum_{\tau=1}^{t} g_t g_t^T$ is matrix of the previous subgradients, and Π_K is the projection of the gradient into the set K. Thanks to this adaption AdaGrad is able to handle noisy feedback, however there is no guarantee that AdaGrad can ensure acceleration, moreover it was unknown whether AdaGrad is able to exploit the smoothness in order to converge faster. In Accelegrad this adaptive learning rate technique is exploited.

The other main idea encapsulated into Accelegrad comes from [2] and is briefly described in the following.

Different from standard GD, Mirror Descent (MD) peeks a uniformly distributed sequences of points $(x_1, \ldots, x_n)$ and it combines them to construct a stronger lower bound to the function f, in particular it is possible to prove that this method guarantees faster convergence rate as the average gradient becomes flatter. This is in contrast with the GD algorithm which provides faster convergence with large gradients. In light of this consideration, [2] propose a linear combination of the two algorithms at each step, in particular following Nesterov, given the gradient step y_k and the mirror step z_k the next step will be given by:

$$x_{k+1} = \alpha z_k + (1 - \alpha) y_k, \tag{4}$$

where α_k is a tuneable parameter. It is possible to prove that this technique provides the same convergence guarantees of the accelerated gradient method of Nesterov in a nicer and simpler way, for this reason Accelegrad incorporates the notion of acceleration taking inspiration from this linear coupling framework.

1.2 Accelegrad algorithm setting

In the Accelegrad paper [3], there is no assumption about the smoothness parameter β (See Def.2.1), but they assumed to be given a bound between some initial point x_0 and a global minimum of f for a given compact convex set K which contains this global minimizer. This distance is bounded by the diameter of K i.e. $D = \max_{x,y \in K} \|x - y\|$. Finally they assume that the function f is G-Lipschitz for which we provide the definition in (Def.2.2).

First, we discuss the offline setting where we have always access to the exact gradients of f, the algorithm is presented in Algo.1. As we previously said, Accelegrad takes inspiration from [2] and linearly couples between two sequences $\{y_t\}_t$ and $\{z_t\}_t$ into $\{x_{t+1}\}_t$. These two sequences are updated in two different manners, the former (y_{t+1}), takes a step starting from x_{t+1}, the latter (z_{t+1}), takes a step starting from z_t. Both sequences are updated with the same step size but for z_{t+1} the gradient is scaled by a factor of α_t and the final result is projected into K. The step size is defined taking inspiration from AdaGrad:

$$\eta_t = \frac{2D}{(G^2 + \sum_{\tau=0}^{t} \alpha_\tau^2 \|g_t\|^2)^2}, \tag{5}$$

where the only difference are the importances weights α_t defined as:

$$\alpha_t = \begin{cases} 1 & 0 \leq t \leq 2 \\ \frac{1}{4}(t+1) & t \geqslant 3 \end{cases}, \tag{6}$$

which increase with t, putting emphasis on recent queries of the gradient.

Algorithm 1: Accelerated Adaptive Gradient Method (AcceleGrad)

Input : #Iterations T, $x_0 \in K$, diameter D, weights $\alpha_{t \in T}$, learning rate η_t

Set $y_0 = x_0 = z_0$

for $t = 1 \dots T$ **do**

  Set $\tau_t = 1/\alpha_t$

  Update:

$$x_{t+1} = \tau_t z_t + (1 - \tau_t) y_t, \quad g_t = \nabla f(x_{t+1})$$

$$z_{t+1} = \Pi_K(z_t - \alpha_t \eta_t g_t)$$

$$y_{t+1} = x_{t+1} - \eta_t g_t$$

end

Output: $\hat{y}_T \propto \sum_{t=0}^{T-1} \alpha_t y_{t+1}$

Given the smooth and the non-smooth case, Accelegrad requires the following number of iterations to get a solution within ϵ respectively:

- $\mathcal{O}(\frac{c}{\epsilon})^{\frac{1}{2}} \quad c = DG + \beta D^2 log(\beta D/G)$
- $\mathcal{O}(\frac{c}{\epsilon})^{\frac{1}{2}} \quad c = DG + \beta D^2 log(\beta D/G)$

The synthetic proofs are provided in ...

2 General definitions

Definition 2.1. A function $f : \mathbb{R}^n \to \mathbb{R}$ is β-smooth if:

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{\beta}{2} \|x - y\|^2, \forall x, y \in \mathbb{R}^n$$

Definition 2.2. f is G-Lipschitz if $\forall w \in \Omega$ we have $\|\nabla f_i(w)\| \leq G$

Definition 2.3. Given a real-valued convex function f with elements $x_1, \dots x_n$ in its domain and positive weights a_i, Jensen's inequality is stated as:

$$f\left(\frac{\sum a_i x_i}{\sum a_i}\right) \leq \frac{\sum a_i f(x_i)}{\sum a_i}$$

3 Appendix with main proofs

Theorem 1. *Assume that f is convex and β-smooth. Let K be a convex set with bounded diameter D, and assume there exists a global minimizer for f in K. Then Algorithm 1 with weights equal to Equation 6 and learning rate as in Equation 5 ensures:*

$$f(\hat{y}_t) - f(x^*) \leq \mathcal{O}\left(\frac{DG + \beta D^2 log(\beta D/G)}{T^2}\right)$$

Since the proof of this Theorem is very long, we provide the most meaningful parts which guides us into deriving the upper bound on the error.

Proof. Our objective is to find a bound of the left-side in order to precisely state the convergence rate of Accelegrad in the smooth case. Given the fact that the algorithm outputs a weighted average of gradients we may apply the Jensen Inequality defined in 2.3 as:

$$f(\hat{y}_t) - f(x^*) \leq \frac{1}{\sum_{t=0}^{T-1} a_i} \sum_{t=0}^{T-1} a_i (f(y_{t+1}) - f(x^*))$$

Given the fact that $\sum_{t=0}^{T-1} a_i \geqslant \Omega(T^2)$ it is sufficient to bound the numerator by a constant in order to get the sublinear converge rate $\mathcal{O}(C/T^2)$ and the proof is concentrated precisely at this point.

Without re-writing the proof of Lemma described in the paper, we can bound our term of interest by:

$$\begin{aligned}\sum_{t=0}^{T-1} a_i (f(y_{t+1}) - f(x^*)) \leq (\alpha_t^2 - \alpha_t)(f(y_t) - f(y_{t+1}) \\ + \frac{\alpha_t^2}{2}\left(\beta_t - \frac{1}{\eta_t}\right)\|y_{t+1} - x_{t+1}\|^2 \\ + \frac{1}{2\eta_t}(\|z_t - z\|^2 - \|z_{t+1} - z\|^2).\end{aligned}$$

Each term of the sum is in turn bounded, providing a more elegant overall bound:

$$\frac{1}{2}\sum_{t=0}^{T-1} a_i (f(y_{t+1}) - f(x^*)) \leq \underbrace{\frac{D^2}{2\eta_{T-1}} - \frac{1}{4}\sum_{t=\tau_*+1}^{T-1} \eta_t \alpha_t^2 \|g_t\|^2}_{(*)} + \underbrace{\frac{\beta}{2}\sum_{t=0}^{\tau_*} \eta_t^2 \alpha_t^2 \|g_t\|^2}_{(**)}.$$

Now by fixing the learning rate equal to the value in Equation.5, we provide a final bound which is constant and we conclude the proof. These steps requires many computations and Lemmas that we skip in order to provide the higher point of view.

The final result is of the form:

$$\frac{1}{2}\sum_{t=0}^{T-1} a_i (f(y_{t+1}) - f(x^*)) \leq DG/4 + 2\beta D^2 + 2\beta D^2(1 + 2log(4\beta D/D)$$

Combining this result with our initial inequality we get:

$$\begin{aligned} f(\hat{y}_t) - f(x^*) &\leq \frac{1}{\sum_{t=0}^{T-1} a_i} \sum_{t=0}^{T-1} a_i (f(y_{t+1}) - f(x^*)) \\ &\leq \frac{DG/2 + 8\beta D^2(1 + log(4\beta D/D)}{T^2/32} \\ &\approx \mathcal{O}\left(\frac{DG + \beta D^2 log(4\beta D/D)}{T^2}\right), \end{aligned}$$

where we used the fact that $\sum_{t=0}^{T-1} a_i \geqslant T^2/32$, which conclude the proof for the smooth case. □

Theorem 2. *Assume that f is convex and G-Lipschitz. Let K be a convex set with bounded diameter D, and assume there exists a global minimizer for f in K. Then Algorithm 1 with weights equal to Equation 6 and learning rate as in Equation 5 ensures:*

$$f(\hat{y}_t) - f(x^*) \leq \mathcal{O}\left(GD\sqrt{logT}/\sqrt{T}\right)$$

Proof. The proof follows the same considerations of the previous one, this time we need to prove that $\sum_{t=0}^{T-1} a_i(f(y_{t+1}) - f(x^*))$ is bounded by $\mathcal{O}(T^{3/2})$. As in the previous case it is possible to prove that:

$$\alpha_t(f(y_{t+1}) - f(x^*)) \leq \eta_t \alpha_t^2 \|g_t\|^2 + \eta_t \alpha_t^2 \|g_t\| + \frac{1}{2\eta_t}(\|z_t^2 - z\| - \|z_{t+1}^2 - z\|)$$

$$+(\alpha_t^2 - \alpha_t)(f(y_t) - f(y_{t+1})).$$

As before, we can bound each term of this sum and the combine the results to conclude our proof. □

Theorem 3. *Assume that f is convex and G-Lipschitz. Let K be a convex set with bounded diameter D, and assume there exists a global minimizer for f in K. Assume that we invole Algorithm 1 with weights equal to Equation 6 and learning rate as in Equation 5 ensures, but this time we provide it with noisy gradient estimates, then our algorithm ensures:*

$$\mathbb{E}[f(\hat{y}_t)] - f(x^*) \leq \mathcal{O}\left(GD\sqrt{logT}/\sqrt{T}\right)$$

Proof. In order to provide the proof of this theorem we assume that upon querying a first order oracle with a point x, we receive a bounded and unbiased gradient estimate $\tilde{g}$ such that $\mathbb{E}[\tilde{g}|x] = \nabla f(x)$ with $\|\tilde{g}\| \leq G$. As in the previous theorems we bound our term of interest by: □

4 Online to Batch conversion

Online learning is the process of answering a sequence of questions given knowledge of the correct answers to the previous ones. Given a sequence of questions $S = (x_1, \ldots, x_t)$, for each one of them sequentially our algorithm makes a prediction and then after the prediction the correct answer is revealed incurring in a loss. The goal of the learner is to minimize the loss also exploiting past information. If there is no correlation between samples, learning is hopeless.

Intuitively, for each sample passed to the learner at stage t, the learner chose an hypothesis h from the hypothesis class H and predicts the label associated with the sample. For this reason we can define a regret function which measure "how sorry" the learner is not to have followed the best predictor h^*. Formally when running on a sequence of T examples:

$$R_T(h^*) = \sum_{t=1}^{T} l(p_t, y_t) - \sum_{t=1}^{T} l(h^*(x_t), y_t),$$

where p_t is the prediction obtained by h at stage t, and the regret w.r.t the hypothesis class H is defined as:

$$R_T(H) := \max_{h^* \in H} R_T(h^*).$$

Usually we look for an algorithm which converge sub-linearly to the lowest possible regret.

What we described in short say that an online learning algorithm wants performance close to the single best hypothesis chosen in hindsight given all the data.

Different from this practice, an online to batch converter needs to construct a model based on the set of modules generated by the online learner, this can be done in different ways but one of the most classical way is to take the average of these models. This conversion is needed because the individual iterates of an online algorithm do not come with individual guarantees.

The standard online to batch conversion algorithm format is shown in Algo.2. In particular since the algorithm outputs an average of the iterates, we can apply Jensen's inequality to show the following:

$$\mathbb{E}[L(\hat{w}) - L(x^*)] \leq \frac{\mathbb{E}[R_T(x^*)]}{T},$$

and as long as the algorithm obtains sublinear regret the left side part of the inequality will approach zero an average.

Algorithm 2: Online to batch conversion

Input : #Iterations T, Params S, Cost function l, Algorithm A

for $t = 1 \dots T$ **do**

  let w_t be the prediction of A, provide the cost function $l(w_t, g_t)$ to A where $\mathbb{E}[g_t] = \nabla F(w_t)$

end

Output: $\hat{w} = \frac{1}{T} \sum_{t=1}^{T} w_t$

In [4] to avoid the guarantees problems of single iterates it is provided a black box "Anytime" Online to Batch conversion algorithm. The "Anytime property" comes from the fact that the last iterate is always a good estimate of x^*. The algorithm is very similar to the classical online to batch previously shown. The key difference is that they evaluate the stochastic gradient oracle at x_t rather than the iterates provided by the algorithm A, furthermore they incorporate the importance weights α_t which are useful to achieve faster convergence rates on smooth losses.

Algorithm 3: Anytime Online to batch conversion

Input : #Iterations T, Online Algorithm A with convex domain D, Non-negative weights α_t with $\alpha_t > 0$

for $t = 1 \dots T$ **do**

  Get $x_t = \frac{\sum_{i=1}^{t} \alpha_i w_i}{\sum_{i=1}^{t} \alpha_i}$

  Play x_t and receive subgradient g_t

  Send $l_t(x) = \langle \alpha_t g_t, x \rangle$ to a as the tth loss

  Get w_{t+1} from A

end

Output: x_T

Different from the standard online to batch conversion, this algorithm is able to achieve the following guarantees for every $x^* \in D$:

$$\mathbb{E}[L(x_T) - L(x^*)] \leq \frac{\mathbb{E}[R_T(x^*)]}{\sum_{t=1}^{T} \alpha_t},$$

in particular this bound is valid also for loss functions with some known non-linearity.

References

[1] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 07 2011.

[2] Z. A. Zhu and L. Orecchia, "A novel, simple interpretation of nesterov's accelerated method as a combination of gradient and mirror descent," *CoRR*, vol. abs/1407.1537, 2014.

[3] K. Y. Levy, A. Yurtsever, and V. Cevher, "Online adaptive methods, universality and acceleration," 2018.

[4] A. Cutkosky, "Anytime online-to-batch conversions, optimism, and acceleration," 2019.