Evaluation
Online learning
links with optimization and games
Université Paris–Saclay

variants of adagrad-norm and application to games

Let $d \geqslant 1$, $\mathcal{X} \subset \mathbb{R}^d$ a nonempty closed convex set, $\gamma, \eta, \mathrm{L} > 0$, $x_0 \in \mathcal{X}$, and $(u_t)_{t\geqslant 0}$ a sequence in $\mathbb{R}^d$. We consider

- *AdaGrad-Norm*, defined as

$$x_{t+1} = \Pi_{\mathcal{X}}\left(x_t + \frac{\gamma}{\sqrt{\sum_{s=0}^t \|u_s\|_2^2}} u_t\right), \qquad t \geqslant 0,$$

  with convention $0/0 = 0$;

- *AdaGrad-DA-Norm*, defined as

$$x_{t+1}^{(\mathrm{DA})} = \Pi_{\mathcal{X}}\left(\frac{\eta}{\sqrt{\mathrm{L}^2 + \sum_{s=0}^t \|u_s\|_2^2}} \sum_{s=0}^t u_t\right), \qquad t \geqslant 0;$$

- *AdaGrad-Hybrid-Norm*, defined as

$$x_{t+1}^{(\mathrm{hyb})} = \Pi_{\mathcal{X}}\left(\eta_{t+1}\left(\frac{x_t^{(\mathrm{hyb})}}{\eta_t} + u_t\right)\right), \qquad t \geqslant 0,$$

where $\eta_t = \eta / \sqrt{L^2 + \sum_{s=0}^{t-1} \|u_s\|_2^2}$.

1) Assume that for all $t \geqslant 0$, $\|u_t\|_2 \leqslant L$. For $T \geqslant 0$ and $x \in \mathcal{X}$, establish an upper bound on the regret

$$\sum_{t=0}^{T} \left\langle u_t, x - x_t^{(\mathrm{DA})} \right\rangle \qquad \left(\text{resp.} \quad \sum_{t=0}^{T} \left\langle u_t, x - x_t^{(\mathrm{hyb})} \right\rangle \right).$$

   Hint. — For $t \geqslant 0$, consider mirror map

$$H_t(x) = \frac{\sqrt{L^2 + \sum_{s=0}^{t-1} \|u_s\|_2^2}}{2\eta} \|x\|_2^2, \qquad x \in \mathbb{R}^d,$$

   and associated regularizer $h_t = H_t + I_{\mathcal{X}}$.

2) Let $m, n \geqslant 1$ be integers, and $A \in \mathbb{R}^{m \times n}$. Apply each of the above algorithms for solving the two-player zero-sum game associated with A and derive corresponding guarantees.

3) Perform numerical experiments in the context of solving two-player zero-sum games and compare the performance of the three above algorithms with RM, RM+ and the exponential weights algorithm. Use the following function to compute the Euclidean projection onto the simplex.

```python
def projection_simplex(y):
    n_features = y.shape[0]
    z = np.sort(y)[::-1]
    cssv = np.cumsum(z) - 1
    ind = np.arange(n_features) + 1
    cond = u - cssv / ind > 0
    rho = ind[cond][-1]
    theta = cssv[cond][-1] / float(rho)
    w = np.maximum(y - theta, 0)
    return w
```

4) Bonus. — Add to the numerical experiments the optimistic variant of each algorithm.

5) Bonus. — Rewrite the above Python function mathematically and prove that it indeed computes the Euclidean projection onto the simplex.

6) Bonus. — Also study the digonal variants of each above algorithm and include them in the numerical experiments.