10-425/625: Introduction to Convex Optimization (Fall 2023)

Lecture 23: Online Gradient Descent / Parallel SGD

Instructor:[1] *Matt Gormley* *December 1, 2023*

23.1 Online Gradient Descent

The *projected online gradient descent* algorithm of Zinkevich (2003) is the online variant of projected gradient descent. That is, suppose we have a constrained problem $\min_{x \in C} \sum_{t=1}^{T} f_t(x)$, but we only get to see one $f_t(\cdot)$ at a time, and must choose a corresponding $x^{(t)}$ in response before the next $f_{t+1}(\cdot)$ arrives.

The update rule uses the projection operator P_C at each iteration to project a gradient step back onto the feasible set:

$$x^{(t+1)} = P_C(x^{(t)} - \eta_t \nabla f_t(x^{(t)})$$
$$\text{where } P_C(x) = \operatorname*{argmin}_{y \in C} \|x - y\|_2$$

If we choose $C = \mathbb{R}^d$, we obtain the *online gradient descent* algorithm.

Regret Analysis For the online gradient descent algorithm, we can obtain the following regret bound.

Theorem 23.1. *Assume our sequence of convex functions $f_1, \ldots, f_T$ has bounded gradients $\|\nabla f_t(x)\|_2 \leq G$, $\forall t, x \in C$ and the diameter of the feasible region C is bounded $\max_{x,y \in C} \|x - y\|_2 \leq D$. If $\eta = \frac{D}{G\sqrt{T}}$, then*

$$R(T) = \sum_{t=1}^{T} f_t(x^t) - \min_{x \in C} \sum_{t=1}^{T} f_t(x) \leq DG\sqrt{T}$$

Proof: [2]

[1]

[2] This is the version of the Zinkevich (2003) proof from Adam Kalai `https://web.archive.org/web/20070416211828/http://people.cs.uchicago.edu/~kalai/online2004/lect1008.pdf`.

Let a minimum of the function be $x^* = \text{argmin}_{x \in C} \sum_{t=1}^{T} f_t(x)$. Now suppose we translate the space such that $x^* = 0$. In 3D, this amounts to sliding the function surface until one of the minima (we assumed convex, not strictly convex) is at the origin. If we run the algorithm on the translated space, the gradients will be identical; only the iterates will have been shifted. So proving a regret bound on the translated space is without loss of generality.

The second term in the regret is now $\sum_{t=1}^{T} f_t(0)$.

Previously, we've shown that projection onto a convex set C is a contraction: $\|P_C(x) - P_C(y)\|_2 \leq \|x - y\|_2$. For our translated C from above we know that the origin is in the feasible set $0 \in C$. So if we let $y = 0$ to be the origin, then:

$$\|P_C(x)\|_2 \leq \|x\|_2.$$

Now, define the potential $\Phi_t = \frac{1}{2\eta}\|x^{(t)}\|_2^2$, which measures how close $x^{(t)}$ is to the optimal $x^* = 0$. Next we bound the difference of subsequent potentials:

$$\begin{aligned}
\Phi_{t+1} - \Phi_t &= \frac{1}{2\eta}(\|x^{(t+1)}\|_2^2 - \|x^{(t)}\|_2^2) \\
&= \frac{1}{2\eta}(\|P_C(x^{(t)} - \eta_t \nabla f_t(x^{(t)})\|_2^2 - \|x^{(t)}\|_2^2) \\
&\leq \frac{1}{2\eta}(\|x^{(t)} - \eta_t \nabla f_t(x^{(t)}\|_2^2 - \|x^{(t)}\|_2^2)
\end{aligned}$$

where the second line is from the algorithm definition and the third because $\|P_C(x)\|_2 \leq \|x\|_2$. Using the fact that $\|a + b\|_2^2 = \|a\|_2^2 + \|b\|_2^2 + 2a^T b$ to expand the first term:

$$\begin{aligned}
\Phi_{t+1} - \Phi_t &\leq \frac{1}{2\eta}(\|x^{(t)} - \eta \nabla f_t(x^{(t)}\|_2^2 - \|x^{(t)}\|_2^2) \\
&= \frac{1}{2\eta}(\|x^{(t)}\|_2^2 + \eta^2 \|\nabla f_t(x^{(t)}\|_2^2 - 2\eta (x^{(t)})^T \nabla f_t(x^{(t)}) - \|x^{(t)}\|_2^2) \\
&= \frac{\eta}{2}\|\nabla f_t(x^{(t)}\|_2^2 - (x^{(t)})^T \nabla f_t(x^{(t)}) \\
&= \frac{\eta}{2} G^2 - (x^{(t)})^T \nabla f_t(x^{(t)})
\end{aligned}$$

The last line is by assumption.

Applying the first-order convexity condition to $x^{(t)}$ and the minimum $x^* = 0$, we obtain:

$$\begin{aligned} f_t(0) &\geq f_t(x^{(t)}) + \nabla f_t(x^{(t)})^T(0 - x^{(t)}) \\ \Rightarrow f_t(x^{(t)}) - f_t(0) &\leq (x^{(t)})^T \nabla f_t(x^{(t)}) \end{aligned}$$

Summing this result with our bound on the potentials shows:

$$f_t(x^{(t)}) - f_t(0) + \Phi_{t+1} - \Phi_t \leq \frac{\eta}{2} G^2$$

This inequality shows that if $f_t(x^{(t)}$ is much larger than $f_t(0)$, then Φ_t must also be much larger than Φ_{t+1}, which means that $x^{(t+1)}$ is much closer to x^* than x^t was.

Now we sum both sides from $t = 1, \ldots, T$:

$$\begin{aligned} \sum_{t=1}^{T} f_t(x^{(t)}) - f_t(0) + \Phi_{t+1} - \Phi_t &\leq \sum_{t=1}^{T} \frac{\eta}{2} G^2 \\ \sum_{t=1}^{T} (f_t(x^{(t)}) - f_t(0)) + (\Phi_{T+1} - \Phi_1) &\leq \frac{\eta T G^2}{2} \end{aligned}$$

Since the potentials are non-negative:

$$\sum_{t=1}^{T} f_t(x^{(t)}) - \sum_{t=1}^{T} f_t(0)) \leq \Phi_1 + \frac{\eta T G^2}{2}$$

By our assumption of bounded diameter, we have that $\Phi_t \leq \frac{1}{2\eta} D^2$. This yields:

$$\begin{aligned} \sum_{t=1}^{T} f_t(x^{(t)}) - \sum_{t=1}^{T} f_t(0)) &\leq \frac{T}{2\eta} D^2 + \frac{\eta T G^2}{2} \\ &\leq DG\sqrt{T} \end{aligned}$$

where the second line is obtained by substituting $\eta = \frac{D}{G\sqrt{T}}$. ■

Corollary 23.2. *The limit of the average regret as T goes to infinity is bounded above by zero:*

$$\lim_{T \to \infty} R(T)/T \leq 0$$

Intuitively, if we were to run the algorithm long enough, the online gradient descent algorithm would do just as well as its offline counterpart.

23.1.1 Other Regret Bounds

AdaGrad Duchi et al. (2011) show that AdaGrad also achieves a $R(T) \in O(\sqrt{dT})$ regret bound where d is the dimensionality of $x \in \mathbb{R}^d$. They further show that in cases where the features are sparse, AdaGrad can obtain a $R(T) \in O(\log d\sqrt{T})$ bound. In this setting AdaGrad can outperform online gradient descent, which because $D = 2\sqrt{d}$ has a regret bound is $R(T) \in O(\sqrt{dT})$.

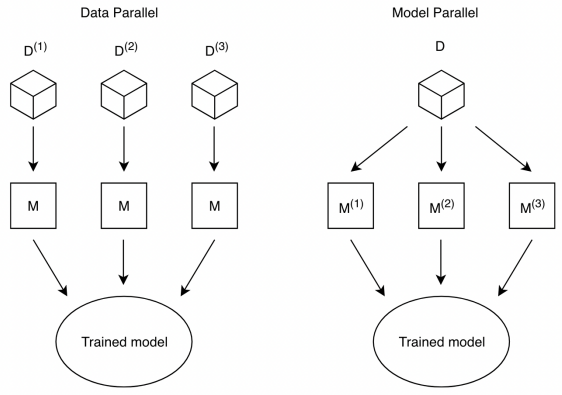
Adam Kingma & Ba (2015) find that Adam enjoys a similar $R(T) \in O(T)$ regret bound.

23.2 Parallel SGD

For many applications, stochastic gradient descent (SGD) is used to minimize empirical risk over a *very* large training dataset. In these cases, parallelization can aid greatly in reducing the wall clock time to convergence. However, parallelization is not without costs such as communication overhead, long wait times for straggling workers, redundant storage of training data. We begin by considering the most straightforward applications of parallelism.

Data Parallelism vs. Model Parallelism Here we focus on *data parallelism* in which the data is distributed.

Modern deep learning architectures are often too large to fit on a single GPU. This setting requires *model parallelism*, in which the model is divided across GPUs.



Comparison of data and model parallelism from Verbraeken et al. (2021).

23.2.1 Distrbuted Synchonous Mini-Batch SGD

Suppose our problem is of the form $\min_x \sum_{i=1}^n f_i(x)$. If the computation cost of computing $f_i(x)$ and $\nabla f_i(x)$ is substantially more than that of sending x between machines and computing the update $x^{(t+1)} = x^{(t)} - \eta \nabla f_t(x)$, then we can achieve trivial parallelism by dividing each mini-batch across m machines.

Algorithm 1 DistributedSyncSGD(m)

Choose initial point $x^0 \in \mathbb{R}^n$
for $t = 1, 2, \ldots, T$ **do**
    On the head node: Send $x^{(t)}$ to each worker node k
    for $k = 1, \ldots, m$ **in parallel do**
        On worker node k:
        Sample minibatch $I_k^{(t)} \subseteq \{1, \ldots, n\}$ of size b
        Compute $g_k^{(t)} = \sum_{i \in I_k^{(t)}} \nabla f_i(x^{(t)})$
        Send $g_k^{(t)}$ to head node
    On the head node: $x^{t+1} = x^t - \eta_t \sum_{k=1}^m g_k^{(t)}$

Although the algorithm is exact (i.e. the parallel version and the local version do identical computation), various problems can arise.

- If one worker node is much slower than the rest, then the computation at each iteration will be bottlenecked by the speed of the slowest node.
- If the time to send gradients $g_k^{(t)}$ and iterates $x^{(t)}$ is comparable to the gradient computation, then the base cost of gradient computation goes up substantially.
- If the dataset is too large to fit on one node, then we will have to substantially increase communication costs by sending data to nodes on demand.