COMS 4995-3: Advanced Algorithms April 19, 2017

Lecture 24 – MWU and Large-scale models

Instructor: *Alex Andoni* Scribes: *Emmanouil Vlatakis*

1 Introduction

Today's lecture is about Multiplicative Weight Update and its application in approximating via LP feasibility problems.

Let's re-introduce again our basic notation:

- We have access to n experts who tell us over the course of T days (on every day) their opinion about an aspect.
- Suppose that f_i^t is representing now a measure the loss of the advice of expert i on day t.
- Suppose that $f_i^t \in [-1, 1]$.
- We define as p^t the vector $(p_1^t, \cdots, p_i^t, \cdots, p_n^t)$ which are the probabilities of following the advice of expert i on day t.
- The number of the mistakes or equivalently the total loss from the advices of the expert i over T days is defined as $m_i^T = \sum_{t=1}^{T} f_i^t$.
- The expected total error is $M^T = \sum_{i=1}^{n} \sum_{t=1}^{T} p_i^t f_i^t$

1.1 Randomizing the choice halfs the error

MWU algorithm:

- $w_i^0 = 1 \ \forall i \in [n]$
- $w_i^{t+1} = w_i^t(1 - \epsilon f_i^t)$
- Choose advice of expert i with probability $p_i^t = \frac{w_i^t}{Q^t}$, where $Q^t = \sum_{i \in [n]} w_i^t$

Theorem 1. $M^T \leq \sum_{t=1}^{T} f_i^t + \frac{\ln n}{\epsilon} + \epsilon T, \ \forall i \in [n]$

Proof. We will upper bound and lower bound as in the previous lecture's strategy the quantity Q^{t+1}

$$\begin{aligned}
Q^{T+1} &= \sum_{i\in[n]} w_i^{T+1} \\
&= \sum_{i\in[n]} w_i^T(1-\epsilon f_i^T) \\
&= \sum_{i\in[n]} w_i^T - \sum_{i\in[n]} w_i^T \epsilon f_i^T \quad (w_i^T = p_i^T \times Q^T) \\
&= \sum_{i\in[n]} w_i^T - \sum_{i\in[n]} p_i^T \times Q^T \times \epsilon f_i^T \\
&= \underbrace{\sum_{i\in[n]} w_i^T}_{Q^T} - \sum_{i\in[n]} p_i^T \times Q^T \times \epsilon f_i^T \\
&= Q^T(1 - \sum_{i\in[n]} p_i^T \epsilon f_i^T) \\
&= Q^T(1-\epsilon\langle p^t, f^t\rangle)
\end{aligned}$$

Using the classical bound $(1-x) \leq e^{-x}$ the last equality implies that:

$$\begin{aligned}
Q^{T+1} &= \sum_{i\in[n]} w_i^{T+1} \\
&\leq Q^T e^{-\epsilon\langle p^t, f^t\rangle} \\
&\leq Q^0 \prod_{t=1}^{T} e^{-\epsilon\langle p^t, f^t\rangle} \\
&\leq \underbrace{Q^0}_{\sum_{i\in[n]} w_i^0 = n} e^{-\sum_{t=1}^T \epsilon\langle p^t, f^t\rangle} \\
&\leq n e^{-\epsilon \sum_{i=1}^n \sum_{t=1}^T p_i^t f_i^t} \\
&\leq n e^{-\epsilon M^T}
\end{aligned}$$

On the other hand we have that $Q^{T+1} \geq w_i^{T+1} \forall i \in [n]$ Following the algorithm we know that :

$$w_i^{T+1} \geq \prod_{t=1}^{T}(1-\epsilon f_i^t) w_i^0 = \prod_{t=1}^{T}(1-\epsilon f_i^t)$$

Given that $\prod_{t=1}^T (1-\epsilon f_i^t) \leq Q^t \leq n e^{-\epsilon M^T} \quad \forall i \in [n]$, we have that:

$$\epsilon M^T \leq \ln n - \sum_{t=1}^{T} \ln(1-\epsilon f_i^t) \quad \forall i \in [n]$$

But $\ln(1-x) \geq -x - x^2 \Rightarrow -\ln(1-x) \leq x + x^2$ so we have that:

$$M^T \leq \ln n/\epsilon + \sum_{t=1}^{T} f_i^t + \epsilon (f_i^t)^2 \quad \forall i \in [n]$$

Additionally, since $f_i^t \in [-1, 1]$, we have that $(f_i^t)^2 \leq 1$.
Therefore:

$$M^T \leq \ln n/\epsilon + \sum_{t=1}^{T} f_i^t + \epsilon \sum_{t=1}^{T} (f_i^t)^2 \leq \frac{\ln n}{\epsilon} + \sum_{t=1}^{T} f_i^t + \epsilon T \quad \forall i \in [n]$$

□

Notice that this result implies immediately that:

$$M^T \leq \frac{\ln n}{\epsilon} + \min(1+\epsilon) m_i^T$$

which is twice better bound than the previous lecture's result.

1.2 Application in LP Feasibility problems

Suppose that we are trying to solve the feasibility constraints problem

$$FeasibilityProblem : \boxed{\textbf{Does exist any } x \in \mathbb{R}^n : Ax \geq b \ ?}$$

We will relax this feasibility problem by the following approximation:

$$ApproximationProblem : \boxed{\begin{cases} \textbf{Find an } x \in \mathbb{R}^n : Ax \geq b - \epsilon \mathbb{1} \\ \textbf{Certify } \nexists x : Ax \geq b \end{cases}}$$

Corollary 2. *From the previous theorem if $T > \ln n/\epsilon^2$ we have that : $M^T \leq \min_{i \in [n]} \sum_{t=1}^{T} f_i^t + 2\epsilon T$*

In order to solve the LP-relaxed problem we will construct a very clever agent mechanism.
Let's assume that we have access to the following oracle:

ORACLE $\mathcal{O}(p \in (\mathbb{R}^+)^m)$

- If there exists $x \in \mathbb{R}^n$ such that $p^\top Ax \geq p^\top b$ and $\max_i |A_i x - b_i| \leq 1$ then outputs a possible x.
- Otherwise, it outputs NULL.

Notice that ideally, if there was a solution to our initial problem we would like the oracle to output something very close to it. The difficulty of that oracle is much less than the initial problem.

Theorem 3. *There is an algorithm A that simulates the oracle $\mathcal{O}$ and use efficiently it $O(\log n/\epsilon^2)$ many times to solve the ApproximationProblem.*

Proof. The proof is just based in the corollary 2 and in the correct simulation of the oracle $\mathcal{O}$. The idea is simple. Every inequality will play the role of an expert. We will run a similar MWU scheme.

Algorithm:

- Start with $w_i^0 = 1 \Big| p_i^0 = \frac{1}{m} \forall i \in [m]$, where m is the number of the different inequalities.
- Call oracle $\mathcal{O}(p^t)$
  - If $\mathcal{O}(p^t) \in \mathbb{R}^n$ then $x^t \leftarrow \mathcal{O}(p^t)$
  - If $\mathcal{O}(p^t) = \text{NULL}$ then it outputs "No Solution".
  - Set $f_i^t = A_i x^t - b_i$. (Notice that the most easily satisfied constraints are less important, so the loss function's value is bigger)
  - $w_i^{t+1} = w_i^t(1 - \epsilon f_i^t)$
  - $p_i^t = \frac{w_i^t}{Q^t}, Q^t = \sum_{i \in [n]} w_i^t$
- Do $T = O(\ln n/\epsilon^2)$ iterations.
- Ouput the time mean value $\tilde{x} = \dfrac{\sum_{t=1}^T x^t}{T}$

To conclude the proof, we will argue about the correctness of the algorithm:

$$\begin{aligned} M^T &= \sum_{i=1}^T \langle p^t, f^t \rangle \\ &= \sum_{i=1}^T \langle p^t, (Ax^t - b) \rangle \\ &= \sum_{i=1}^T \langle p^t, Ax^t \rangle - \sum_{i=1}^T \langle p^t, b \rangle \\ &= \sum_{i=1}^T \Big(\langle p^t, Ax^t \rangle - \sum_{i=1}^T \langle p^t, b \rangle \Big) \\ &\geq 0 \end{aligned}$$

The last inequality can be derived by the fact that $\big(\langle p^t, Ax^t \rangle \geq \langle p^t, b \rangle\big)$ because of the construction of the oracle.

Again using the Lower.bound $\leq$ Upper.Bound we have that: $\begin{cases} 0 \leq M^T \\ M^T \leq \min_{i \in [n]} \sum_{t=1}^T f_i^t + 2\epsilon T \end{cases} \Rightarrow$ $\min_{i \in [n]} \sum_{t=1}^T f_i^t + 2\epsilon T \geq 0 \Rightarrow \min_{i \in [n]} \sum_{t=1}^T (A_i x^t - b_i) + 2\epsilon T \geq 0 \Rightarrow \frac{1}{T}\Big(\sum_{t=1}^T x^t \Big) \geq b_i - 2\epsilon \ \forall i \in [n]$

Notice that $\tilde{x}$ resolved indeed the approximate problem. In addition notice that the property $f_i^t \in [-1, 1]$, is guaranteed as $\max_{i \in [n]} |f_i^t| \leq 1$. □

2 Large Scale models

In this section we study the Input/Output (I/O) complexity of large-scale problems arising e.g. in the areas of database systems, geographic information systems, VLSI design systems and computer graphics, and design I/O-efficient algorithms for them.

Traditionally when designing computer programs people have focused on the minimization of the internal computation time and ignored the time spent on Input/Output (I/O).Theoretically one of the most commonly used machine models when designing algorithms is the Random Access Machine (RAM).

One main feature of the RAM model is that its memory consists of an (infinite) array, and that any entry in the array can be accessed at the same (constant) cost. However, in practice there is a huge difference in access time of fast internal memory (usually so-called "cache memory") and slower external memory such as disks or main RAM.

While typical access time of cache is measured in nano seconds, a typical access time of a disk or of the main memory is on the order of milli seconds. So roughly speaking there is a factor of a million in difference in the access time of internal and external memory, and therefore the assumption that every memory cell can be accessed at the same cost is questionable, to say the least.

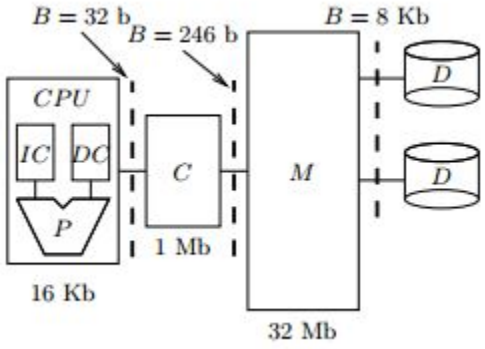

Figure 1: A real machine with typical memory and block sizes

Definition 4 (The I/O Mode)**.** *So we will modify the way that we count the complexity of our model following an external memory model (I/O model) in 2 levels.*

(0th layer) *CPU contains a register file.*

(1st layer) *If CPU cannot find the demanded information in the register file, it exchanges blocks extremely fast with a cache memory C.*

(2nd layer) *If the demanded memory block is not located there then CPU seeks it into the external memory M .*

NOTE$_1$ *Each memory element (register file,cache memory,external memory) is organized in cache lines, of length B (the so-called "block size").*

NOTE$_2$ *If we need the information of address addr from cache and this address belongs to block $\mathcal{B} = [LeftAddr, RightAddr]$, then we need to transfer the whole block from the external memory.*

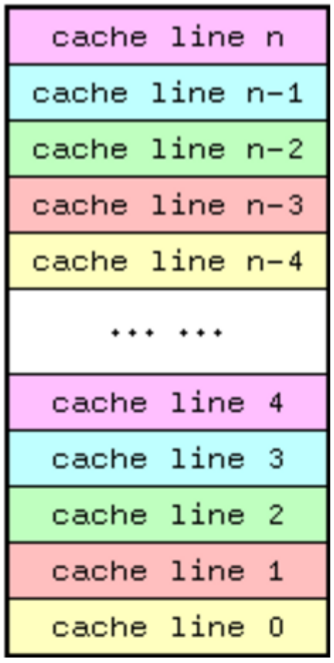

Figure 2: Cache line Organization

Example 5 (Cache Line Organization)**.** *Suppose that cache memory has size* $|C| = 1Mb$*, with block size* 32 *bits=4 bytes. Then, the cache memory contains* $|C|/4$ *cache lines, or equivalently* $CacheLines = |C|/|BlockSize| = 2^{20}/4 = 2^{18}$ *cache lines.*

Example 6 (Block Transfer)**.** *If we need the information of address=0004 and the block size is 8 we have to bring the whole block, the whole cache line [0000,0007].*

Definition 7. *There are many different ways to count the complexity of a model. However, the main bottleneck is the communication between 1st and 2nd layer.*

Cost(I/O Time) = # times need to bring a cache line into cache from the external memory (or vice versa)

Example 8 (Searching in Unsorted Array)**.** *Task: Scanning an array of* N *numbers in an unsorted array and searching an element* e *is possible in* $\mathcal{O}(\lceil N/B \rceil)$ I/O *calls..*

Proof. Firstly notice that in the worst case scenario the array may not be aligned into the block. That means that the first element of the array may not be simultaneously the first element of some block.

If $N \leq B$ then after at most 2 transfers from RAM to cache memory we can find the element e.

If $N > B$ then using an adversary argument we can conclude that maybe we have to search the whole array. Therefore we may have to transfer the whole array to the cache memory, so we may transfer at most $\lceil N/B \rceil + 1$ blocks. □

Example 9 (Random Access)**.** *If we have access to* N *random elements of an array then with high probability at each access, we will need access to a different block. Thus* N *random accesses take* N I/O *calls.*

Example 10 (Binary Search in a sorted array)**.** *Suppose that we are running a binary search algorithm in a sorted array for an element x. There is an I/O call process that need $\mathcal{O}\Big(\log(N/B)\Big)$ I/O calls.*

Algorithm 1 Binary Search Algorithm

Proof.
```
1:  procedure BinarySearch(l, r, x)        ▷ The element x belongs in the range [A[l], A[r]]
2:      if l > r then
3:          return NIL                      ▷ The element x is not found.
4:      pivot ← ⌊(l+r)/2⌋
5:      if x = A[pivot] then
6:          return pivot                    ▷ The element x is found.
7:      else if x < A[pivot] then
8:          return BinarySearch(l,pivot-1,x)
9:      else
10:         return BinarySearch(pivot+1,r,x)
```

Notice that while $r - l >> B$, it is almost sure that our process will need to bring a new block, so we will need to execute a new I/O call. On the other hand when $r - l \approx B$, we will continue our binary search inside the last block. Additionally, at each iteration, the searching interval halfs. Then if we execute totally t I/O calls, we have that: $N/2^t \leq B \Rightarrow t \geq \ln(\lceil N/B \rceil)$. □

Suppose that $B << \sqrt{N}$ then the previous algorithm executes $\frac{1}{2}\ln(N)$ I/O calls.

3 Next Time

Claim 11. *It is possible to binary search in a sorted table executing $\mathcal{O}(\log_B N) = \frac{\ln N}{\ln B}$ I/O calls using B-trees.*