Gradient Descent
& Reductions
Problem Set 9 – CS6515 (Spring 2025)

- This problem set is due on **Thursday April 10th**.
- Submission is via Gradescope.
- Your solution must be a typed pdf (e.g., via LaTeX, Markdown, etc. Anything that allows you to type math notation) – no handwritten solutions.
- Please try to make your solutions as concise and readable as possible. Most problems will have solutions that are no more than a page long. Consider using bullet points and adding space to break up large paragraphs into smaller chunks.
- There are 3 problems. Each problem is graded with 20p. **There is +1p bonus per problem** for stating (i) how long it took you to solve that problem, and (ii) how long it took you to type the answer.

25 Finding A Step Size

Let f be an L-smooth convex function. In class we studied gradient descent with fixed step size $\eta \leq 1/L$. To pick η, we need to know L. The proof relied on the following lemma from class:
"For stepsize $\eta \leq 1/L$ we have $f(x^{t+1}) \leq f(x^t) - \frac{\eta}{2}\|\nabla f(x^t)\|^2$." So the best choice would be $\eta = 1/L$ as then the upper bound is as small as possible.

Can we get a similar bound without knowing L? In each iteration we perform $x^{t+1} \leftarrow x^t - \eta_t \nabla f(x^t)$ where η_t must be computed in some way.

Problem:

1. Design an algorithm that, given x^t and $\nabla f(x^t)$, returns a η_t with $f(x^{t+1}) \leq f(x^t) - \frac{1}{4L}\|\nabla f(x^t)\|^2$.
2. Prove correctness.
3. Analyze the time complexity.

You may assume that computing $f(x)$ takes $O(1)$ time and your algorithm is allowed to compute values of f.
For the complexity, it's fine if your algorithm's time complexity depends on $|\log \eta_t|$, where η_t is the value returned by your algorithm.

Remark: This implies $f(x^t) - f(x^*) \leq \frac{\|x^0 - x^*\|^2}{tL}$. So even though we do not know L, we can still guarantee $O(1/\epsilon)$ rate of convergence.

Hint: Repeated doubling/halving.
How much does the function value $f(x^{t+1})$ decrease for $\eta \in [1/(2L), 1/L]$? Your returned η_t might not be from that interval, but can you argue that your returned η_t is at least as good?

26 Non-Convex Function

In class we assumed function f is convex. We now want to consider the non-convex case. We want to show that for L-smooth f, after t iterations with step size $\eta \leq 1/L$ we can find a point x' with

$$\|\nabla f(x')\| \leq \sqrt{\frac{2}{\eta \cdot t}(f(x^0) - f(x^*))}.$$

Note that for a local optimum we have $\nabla f(x) = 0$, so a small norm $\|\nabla f(x')\|$ indicates that we are close to a local optimum or saddle point.

We split the proof into 2 subproblems. Each subproblem can be solved in a few lines of calculation/algebra.

Problem:

1. Show $\sum_{k=0}^{t} \|\nabla f(x^k)\|^2 \leq \frac{2}{\eta}(f(x^0) - f(x^*))$ (Hint: In class we have proven $f(x^{t+1}) \leq f(x^t) - \frac{\eta}{2}\|\nabla f(x^t)\|^2$, which implies $\frac{\eta}{2}\|\nabla f(x^t)\|^2 \leq ...$)
2. Show $\min_{k=0...t} \|\nabla f(x^k)\| \leq \sqrt{\frac{2}{\eta \cdot t}(f(x^0) - f(x^*))}$. (Hint: Minimum vs Average.)

where x^* is the global optimum $f(x^*) = \min_x f(x)$.

You are allowed to use subproblem 1 to solve subproblems 2, even if you did not prove 1.

27 Reduction: OV to Regular Expression

We often use regular expressions to process strings. E.g., we might type something like '`str.match("a(.+)(b|c)")`" which checks if `str` is a string that starts with `a`, followed by any number of characters (but at least one character), and then ending with either `b` or `c`.

When we use regular expressions in our favorite programming language, how quickly can it match regular expressions? We here want to argue via reduction that there is no algorithm/procedure for matching regular expressions faster than $O(mn)$ time, where n is the string length and m is the length of the regular expression.

Regular Expression Let's quickly define the set of valid regular expressions for this exercise.

- `.` (dot) matches is any character
- `+` (plus) means that the previous group must occur at least once, but could also occur more often. E.g. `f(ab)+` matches `fab` and `fabab` and `fababab` etc, but it does not match `f`.
- `*` (star) means that the previous group could occur any number of times, including 0 times. E.g. `c(ab)*` matches `c` and `cab` and `cabab` and `cababab` etc.
- `|` (pipe) is used as an "or." E.g., `f((ab)|(cd))` matches `fab` and `fcd`, because `f` is followed by an `ab` or `cd`.

Let us assume matching a string of length n and regular expression of length m takes time $T(n, m)$.

Problem: Give an algorithm that solves the OV-problem in $T(O(nd), O(nd)) + O(nd)$ time. Give algorithm description and complexity analysis. No proof of correctness required.

(OV-problem: Given $u_1, ..., u_n, w_1, ..., w_n \in \{0,1\}^d$, return true/false if there exists a pair with $u_i^\top w_j = 0$.)

Remark: We are basically asking you to solve the OV-problem via regular expressions. Do not compute pairwise dot-products $u_i^\top w_j$. Instead, given vectors $v_1, ..., v_n \in \{0,1\}^d$ construct a regexp `exp` of length $O(nd)$, and given vectors $w_1, ..., w_n \in \{0,1\}^d$ construct a string `str` of length $O(nd)$. Then `str.match(exp)` should return true if and only if there is u_i, w_j with $u_i^\top w_j = 0$ (i.e., a pair of orthogonal vectors).

Complexity Implication: This means, if there was an algorithm that could match string and regexp in time faster than $O(nm)$ (let's say $T(n, m) = O(n^{1-\epsilon}m)$), then we could solve OV in time $O(T(nd, nd) + nd) = O((nd)^{2-\epsilon} + nd) = O(n^{2-\epsilon}\,\mathrm{poly}(d))$. This would contradict the OV-conjecture. So unless the OV-conjecture is wrong, no algorithm can match strings and regexp in time faster than $O(mn)$.

Hint: It can help to start with only one vector $v \in \{0,1\}^d$. Give a regexp of length d that matches any 0/1-string that is orthogonal to v. E.g., for $v = (1,0,0)$ we want a single regexp of length 3 that matches any of 000, 001, 010, and 011.

Next, consider a string that contains many vectors, separated by character 2. Eg for vectors $w_1 = (1,0,1)$, $w_2 = (0,1,1)$, $w_3 = (1,1,0)$ consider the string s=2101201121102. Given some $v \in \{0,1\}^d$, what should the regexp look like, so it matches string s if and only if any of the vector w_i is orthogonal to v?

How do you combine the length $O(d)$ regexp for each v_i into one long regexp of length $O(nd)$?

27.1 Programming Option

Input There are code templates on Piazza that handle reading the input. The input will be of the following format. The first line is n and d, separated by space. The next $2n$ lines are each a 0/1-string of length d. Each of the first n lines represents a vector $u_1, ..., u_n \in \{0,1\}^d$, and each of the next n lines represents a vector $w_1, ..., w_n \in \{0,1\}^d$. Example input:

```
2 2
10
01
11
01
```

This represents $u_1 = (1,0), u_2 = (0,1)$ and $w_1 = (1,1), w_2 = (0,1)$.

Output Your program must return/print 2 lines: The first line represents a regular expression, the second line represents a string. The string should match the regular expression if and only if there is a pair of orthogonal vectors.

Basically, after reading the input, your algorithm should solve the OV-problem via regular expression. The call to "`str.match(regexp)`" is done by the autograder. So your program only needs to construct the string and regexp.

There is no example output, since figuring out what the string and regexp should look like is part of the exercise.

The test cases are public on Piazza. The provided output files only contain a 0 or 1 depending on whether there is an orthogonal pair. It is not an example output for your algorithm! If you want to test your algorithm locally, use `import re` and call `re.match(your_regexp, your_str)`.

Grades: Gradescope will automatically test your submission. You can resubmit as often as you want (until the due date) and you can see on gradescope how many test cases you currently passed. There are no partial points for passing part of the test cases, because even very wrong solutions can pass some test cases. For partial points, submit a theory/pdf solution instead. While your submission must pass all test cases on gradescope, the final pass/fail decision is made by a TA. (This is to make sure your submission is actually solving the problem described above. For example, hardcoding the output is obviously not a valid solution.) We need this rule because all test cases are public (on Piazza) to help you test/debug your program.