

CPSC 540 Assignment 2 (due January 31 at midnight)

The assignment instructions are the same as for the previous assignment, but for this assignment you can work in groups of 1-3. However, please only hand in one assignment for the group.

1. Name(s):
2. Student ID(s):

1 Calculation Questions

1.1 Convexity

Show that the following functions are convex, by only using one of the definitions of convexity (i.e., without using the “operations that preserve convexity” or using convexity results stated in class):¹

1. Upper bound on second-order Taylor expansion: $f(v) = f(w) + \nabla f(w)^T(v - w) + \frac{\lambda}{2}\|v - w\|^2$.
2. Probit regression: $f(w) = \sum_{i=1}^n -\log p(y_i|w, x_i)$ (where the probability is defined as in the last assignment).
3. Maximum function: $f(w) = \max_{j \in \{1, 2, \dots, d\}} w_j$.

Hint: Max are is not differentiable in general, so you cannot use the Hessian for the last one. For the second one, you can assume that $PDF(z) \geq -z \cdot CDF(z)$ (where $PDF(z)$ is the PDF of a standard normal and $CDF(z)$ is the CDF).

Show that the following functions are convex (you can use results from class and operations that preserve convexity if they help):

4. 1-class SVMs: $f(w, w_0) = \sum_{i=1}^N [\max\{0, w_0 - w^T x_i\} - w_0] + \frac{\lambda}{2}\|w\|_2^2$, where $\lambda \geq 0$ and w_0 is a variable.
5. Mixed-norm regularization: $f(w) = \|w\|_{p,q} = \left(\sum_{g \in \mathcal{G}} \|w_g\|_q^p\right)^{\frac{1}{p}}$ (for $p \geq 1$ and $q \geq 1$).
6. Minimum entropy over pairs of variables: $f(w) = \max_{\{i,j \mid i \neq j\}} \{w_i \log w_i + w_j \log w_j\}$ subject to $w_i > 0$ for all i .

1.2 Convergence of Gradient Descent

For these questions it will be helpful to use the “convexity inequalities” notes posted on the webpage.

1. In class we showed that if ∇f is L -Lipschitz continuous and f is strongly-convex, then with a step-size of $\alpha_k = 1/L$ gradient descent has a convergence rate of

$$f(w^k) - f(w^*) = O(\rho^k).$$

¹That C^0 convex functions are below their chords, that C^1 convex functions are above their tangents, or that C^2 convex functions have a positive semidefinite Hessian.

Show that under these assumptions that a convergence rate of $O(\rho^k)$ in terms of the function values implies that the iterations have a convergence rate of

$$\|w^k - w^*\| = O(\rho^{k/2}).$$

2. A basic variation on the Armijo line-search is to set the step-size α_k to be $\alpha_k = (0.5)^p$, where p is the smallest constant such that

$$f(w^k - \alpha_k \nabla f(w^k)) \leq f(w^k) - \frac{\alpha_k}{2} \|\nabla f(w^k)\|^2.$$

Show that if ∇f is Lipschitz-continuous and f is bounded below, that setting α_k in this way means gradient descent requires at most $t = O(1/\epsilon)$ iterations to find at least one iteration k with $\|\nabla f(w^k)\|^2 \leq \epsilon$. Hint: you'll first want to figure out a progress bound of the form

$$f(w^{k+1}) \leq f(w^k) - \gamma \|\nabla f(w^k)\|^2,$$

for some constant γ that holds for all iterations. It may also help to recognize that if a function is L -Lipschitz continuous that it is also L' -Lipschitz continuous for any $L' \geq L$.

3. Show that if ∇f is Lipschitz-continuous and f is convex (but not necessarily strongly-convex or PL), that if we run gradient descent for k iterations with $\alpha_k = 1/L$ we have

$$f(w^k) - f(w^*) = O(1/k).$$

Hint: you will have to use one the C^1 definition of convexity in our usual progress bound coming from Lipschitz-continuity of the gradient. You can try to turn this into a telescoping sum in terms of $\|w^k - w^*\|^2$ (which involves “completing the square”). Finally, note that our usual progress bound also gives that $f(w^{k+1}) \leq f(w^k)$ for all k .

1.3 Beyond Gradient Descent

1. We can write the proximal-gradient update as

$$\begin{aligned} w^{k+\frac{1}{2}} &= w^k - \alpha_k \nabla f(w^k) \\ w^{k+1} &= \operatorname{argmin}_{v \in \mathbb{R}^d} \left\{ \frac{1}{2} \|v - w^{k+\frac{1}{2}}\|^2 + \alpha_k r(v) \right\}. \end{aligned}$$

Show that this is equivalent to setting

$$w^{k+1} \in \operatorname{argmin}_{v \in \mathbb{R}^d} \left\{ f(w^k) + \nabla f(w^k)^T (v - w^k) + \frac{1}{2\alpha_k} \|v - w^k\|^2 + r(v) \right\}.$$

2. In class we showed that if ∇f is coordinate-wise L -Lipschitz and satisfies PL then randomized coordinate optimization with a step-size of $1/L$ satisfies

$$\mathbb{E}[f(w^{k+1}) - f^*] \leq \left(1 - \frac{\mu}{Ld}\right) [f(w^k) - f^*].$$

Consider a C^2 function satisfying PL where coordinate i has its own Lipschitz constant L_j ,

$$\nabla_{jj}^2 f(w) \leq L_j.$$

Consider using a step-size of $1/L_{j_k}$ and sampling j_k proportional to the L_j , $p(j_k = j) = \frac{L_j}{\sum_{j'} L_{j'}}$. Show that this gives a faster convergence rate. Hint: the previous result corresponds to using $L = \max_j \{L_j\}$.

3. Consider an SVM problem where the x^i are *sparse*. In the bonus slides, we show how to efficiently apply stochastic subgradient methods in this setting by using the representation $w^k = \beta^k v^k$. Now consider a case where we know an L2 ball that contains the optimal solution. In other words, we know a τ such that $\|w^*\| \leq \tau$. If τ is small enough, we can use a *projected* stochastic subgradient method (projecting onto the L2-ball):

$$w^{k+\frac{1}{2}} = w^k - \alpha_k g^k - \alpha_k \lambda w^k$$

$$w^{k+1} = \begin{cases} w^{k+\frac{1}{2}} & \text{if } \|w^{k+\frac{1}{2}}\| \leq \tau \\ \frac{\tau}{\|w^{k+\frac{1}{2}}\|} w^{k+\frac{1}{2}} & \text{if } \|w^{k+\frac{1}{2}}\| > \tau \end{cases}$$

By constraining the w^k to a smaller set, this can sometimes dramatically improve the performance in the early iterations.² However, the projection operator is a full-vector operation so this would substantially slow down the runtime of the method. [Derive a recursion that implements this algorithm without using full-vector operations.](#) Hint: you may need to track more information than β^k and v^k .

2 Computation Questions

2.1 Proximal-Gradient

If you run the demo `example_group.jl`, it will load a dataset and fit a multi-class logistic regression (softmax) classifier. This dataset is actually *linearly-separable*, so there exists a set of weights W that can perfectly classify the training data (though it may be difficult to find a W that perfectly classifies the validation data). However, 90% of the columns of X are irrelevant. Because of this issue, when you run the demo you find that the training error is 0 while the test error is something like 0.2980.

1. Write a new function, `logRegSoftmaxL2`, that fits a multi-class logistic regression model with L2-regularization (this only involves modifying the objective function). [Hand in the modified loss function and report the validation error achieved with \$\lambda = 10\$ \(which is the best value among powers of 10\).](#) [Also report the number of non-zero parameters in the model and the number of original features that the model uses.](#)
2. While L2-regularization reduces overfitting a bit, it still uses all the variables even though 90% of them are irrelevant. In situations like this, L1-regularization may be more suitable. Write a new function, `logRegSoftmaxL1`, that fits a multi-class logistic regression model with L1-regularization. You can use the function `findMinL1`, which minimizes the sum of a differentiable function and an L1-regularization term. [Report the number of non-zero parameters in the model and the number of original features that the model uses.](#)
3. L1-regularization achieves sparsity in the *model parameters*, but in this dataset it's actually the *original features* that are irrelevant. We can encourage sparsity in the original features by using *group* L1-regularization. Write a new function, `proxGradGroupL1`, to allow (disjoint) *group* L1-regularization. Use this within a new function, `softmaxClassifierGL1`, to fit a group L1-regularized multi-class logistic regression model (where *rows* of W are grouped together and we use the L2-norm of the groups). [Hand in both modified functions \(`logRegSoftmaxGL1` and `proxGradGroupL1`\) and report the validation error achieved with \$\lambda = 10\$.](#) [Also report the number of non-zero parameters in the model and the number of original features that the model uses.](#)

²We can obtain a value for τ for any L2-regularized problem where the loss is bounded below, by using that $f(w^*) + \frac{\lambda}{2} \|w^*\|^2 \leq f(0)$ and replacing $f(w^*)$ with its lower bound (which can be 0 for SVMs).

2.2 Coordinate Optimization

The function `example_CD.jl` loads a dataset and tries to fit an L2-regularized logistic regression model using coordinate optimization. Unfortunately, if we use L_f as the Lipschitz constant of ∇f , the runtime of this procedure is $O(d^3 + nd^2 \frac{L_f}{\mu} \log(1/\epsilon))$. This comes from spending $O(d^3)$ computing L_f , having an iteration cost of $O(nd)$, and requiring a $O(d \frac{L_f}{\mu} \log(1/\epsilon))$ iterations. This non-ideal runtime is also reflected in practice: the algorithm's iterations are relatively slow and even after 500 “passes” through the data it isn't particularly close to the optimal function value.

1. Modify this code so that the runtime of the algorithm is $O(nd \frac{L_c}{\mu} \log(1/\epsilon))$, where L_c is the Lipschitz constant of *all* partial derivatives $\nabla_i f$. You can do this by modifying the iterations so they have a cost $O(n)$ instead of $O(nd)$, and instead of using a step-size of $1/L_f$ they use a step-size of $1/L_c$ (which is given by $\frac{1}{4} \max_j \{\|x_j\|^2\} + \lambda$, where x_j is column j of the matrix X). **Hand in your code and report the final function value and total time.**
2. To further improve the performance, make a new version of the code which samples the variable to update j_t proportional to the individual Lipschitz constants L_j of the coordinates, and uses a step-size of $1/L_{j_t}$. You can use the function `sampleDiscrete` (in `misc.jl`) to sample from a discrete distribution given the probability mass function. **Hand in your code, and report the final function value as well as the number of passes.**
3. **Report the number of passes the algorithm takes as well as the final function value** if you use *uniform sampling* but use a step-size of $1/L_{j_t}$.
4. Suppose that when we use a step-size of $1/L_{j_t}$, we see that uniform sampling outperforms Lipschitz sampling. **Why could this be consistent with the bounds we've shown?**

2.3 Stochastic Gradient

If you run the demo `example_SG.jl`, it will load a dataset and try to fit an L2-regularized logistic regression model using 10 “passes” of stochastic gradient using the step-size of $\alpha_t = 1/\lambda t$ that is suggested in many theory papers. Note that in other high-level languages (like R/Matlab/Python) this demo would run really slowly so you would need to write the inner loop in a low-level language like C, but in Julia you can directly write the stochastic gradient code and have it run fast.

Unfortunately, despite Julia making this code run fast compared to other high-level languages, the performance of this stochastic gradient method is atrocious. It often goes to areas of the parameter space where the objective function overflows and the final value is usually in the range of something like $6.5 - 7.5 \times 10^4$. This is quite far from the solution of 2.7068×10^4 and is even worse than just choosing $w = 0$ which gives 3.5×10^4 . (This is unlike gradient descent and coordinate optimization, which never increase the objective function if your step-size is small enough.)

1. Although $\alpha_t = 1/\lambda t$ gives the best possible convergence rate in the worst case, in practice it's typically horrible (as we're not usually optimizing the hardest possible λ -strongly convex function). Experiment with different choices of step-size sequence to see if you can get better performance. **Report the step-size sequence that you found gave the best performance, and the objective function value obtained by this strategy for one run.**
2. Besides tuning the step-size, another strategy that often improves the performance is using a (possibly-weighted) average of the iterations w^t . Explore whether this strategy can improve performance. **Report the performance with an averaging strategy, and the objective function value obtained by this strategy for one run.** (Note that the best step-size sequence with averaging might be different than without averaging.)

3. A popular variation on stochastic is AdaGrad, which uses the iteration

$$w^{k+1} = w^k - \alpha_k D_k \nabla f(w^k),$$

where the element in position (j, j) of the diagonal matrix D_k is given by $1/\sqrt{\delta + \sum_{k'=0}^k (\nabla_j f_{i_{k'}}(w^{k'}))^2}$. Here, i_k is the example i selected on iteration k and ∇_j denotes element j of the gradient (and in AdaGrad we typically don't average the steps). Implement this algorithm and experiment with the tuning parameters α_t and δ . **Hand in your code as well as the best step-size sequence you found and again report the performance for one run.**

4. Implement the SAG algorithm with a step-size of $1/L$, where L is the maximum Lipschitz constant across the training examples ($L = \frac{1}{4} \max_i \{\|x^i\|^2\} + \lambda$). **Hand in your code and again report the performance for one run.**

3 Very-Short Answer Questions

Give a short and concise 1-sentence answer to the below questions.

1. What is a situation where we can violate the golden rule on our test set, and still have it be a reasonable approximation of test error?
2. Do convex functions necessarily have a minima?
3. What is the purpose of the descent lemma?
4. Why did we show that the gradient norm converges to 0 when we run gradient descent? (As opposed to showing that gradient descent finds the global optimum.)
5. What is the relationship between the projected-gradient method and the proximal-gradient method?
6. Why do we prefer the proximal-gradient method over the subgradient method?
7. Why did we say in 340 that regularizing by the L2-norm doesn't give sparsity, and now we're saying we can use the L2-norm to give sparse solutions?
8. What is an example of a pattern that can be enforced with structured sparsity?
9. Under what condition do we prefer to use coordinate optimization over gradient descent?
10. For a machine learning objective that is the sum over n training examples, why should we never use the subgradient method?
11. What is the advantage of SAG over stochastic gradient methods without a memory?