Stony Brook University
CSE592 – Convex Optimization – Spring 18
Homework 4, Due: April, 24, 2018, 11:59PM

April 17, 2018

Instructions

- The homework is due on April 24, 2018. Anything that is received after the deadline will not be considered.
- The write-up **must** be prepared in Latex or Word and converted to pdf. No scanned hand-written notes!
- We can use any Latex class you like, just report question number and your answer.
- If the question requires you to implement a Python function, you must also submit a file with the implementation. Make sure it is sufficiently well documented that the TAs can understand what it is happening.

1 Mirror Descent

We will derive the update of Mirror Descent when using the distance generating function $\Psi : \mathbb{R}^d \to \mathbb{R}$ defined as $\Psi(x) = \frac{1}{2(p-1)}\|x\|_p^2$, where $1 < p \leq 2$. This distance generating function is 1-strongly convex with respect to $\|\cdot\|_p$. Changing p in $(1, 2]$, this will give us a family of algorithms, that includes subgradient descent. Remember that the p-norm of a vector $x \in \mathbb{R}^d$ is defined as $(\sum_{i=1}^d |x_i|^p)^{\frac{1}{p}}$. The update of Mirror Descent is $x_{t+1} = \nabla\Psi^*(\nabla\Psi(x_t) - \eta_t g_t)$. So, we will break this update in several questions.

1. Calculate the Fenchel conjugate of $\Psi(x)$.
2. Calculate the gradient of $\Psi(x)$.
3. Calculate the gradient of $\Psi^*(x)$.
4. Put all togheter and write the update rule for Mirror Descent. Do simplify what you get, in order to have something “nice”.

2 Programming Exercise

In this exercise, we will implement subgradient descent and AdaGrad with full and stochastic subgradients. The included archive contains partial Python code. In order to compile and run the code, you need to install Python3 and you will need NumPy package for the future exercises. One easy way to do set up both of them is to install Anaconda which is a free Python distribution that includes many Python packages for science, math, engineering, data analysis.

We will use subgradient descent to train a linear binary classifer using the hinge loss function. The training data is a subset of the data from the HIGGS boson dataset. The main file will load the data, train the classifiers using SGD, subgradient descent and Adam (with stochastic subgradients). Then, it will plot how the value of the objective function varies during the optimization with respect to the number of iterations and with respect to the wall-clock time.

The included archive contains partial python code, which you must complete. Areas that you will fill in are marked with "TODO" comments. **For this section, you should turn in *only* the files algorithms.py and hw4_functions.py.**

As usual, do experiments with the algorithms to get a feeling of how they behave.

2.1 Algorithms

Complete the implementation of the subgradient descent. Complete the implementation of the subgradient descent method in `algorithms.py`. Use step sizes defined as $\eta_t = \frac{\alpha}{\sqrt{t}}$, where α is the initial step size that is passed to the function as an argument. There is no stopping criterion, so stop the algorithm when the maximum number of iterations is reached and return the last x_t (all the previous ones are returned by the code anyway).

Complete the implementation of AdaGrad. Complete the implementation of the AdaGrad in `algorithms.py`. Use step sizes for coordinate j defined as $\eta_{t,j} = \frac{\alpha}{\sqrt{\epsilon + \sum_{i=1}^{t} g_{i,j}^2}}$, where α is the initial step size and ϵ is the initial sum of the squared gradients, both passed to the function as arguments. There is no stopping criterion, so stop the algorithm when the maximum number of iterations is reached and return the last x_t (all the previous ones are returned by the code anyway). Use matrix/vector operations in the update over the coordinates, **no for/loops!**

2.2 Oracles for the hinge loss

The file `hw4_functions.py` contains the functions to calculate the value of the 'full' and stochastic subgradients of an average of hinge losses. Complete the functions, so that the value returned by `svm_objective_function` and `svm_objective_function_stochastic` with order=0 corresponds to

$$\frac{1}{n}\sum_{i=1}^{n} \max(1 - y_i w^\top x_i, 0)$$

where the matrix x is in the parameter 'features', w is in 'w', y is in 'labels'.

Also, `svm_objective_function` called with order=1 should return a ‘full’ subgradient and value=inf, while `svm_objective_function_stochastic` should return a stochastic gradient calculated over ’minibatch_size’ number of random samples and value=inf. The value=inf for the order=1 is needed to avoid wasting time in the subgradient computation. In fact, remember that in subgradient descent we don’t need the function value during the optimization, but only in the end if we want to select the argmin of $f(x_t)$.

2.3 Implementation Tips

For simplicity, I suggest to define matrix and vectors as NumPy matrices, so that you can use the “*” for matrix-vector multiplication, “.T” for the transpose, and $``.I''$ for the matrix inverse.