Homework 3

Optimization, 18-660/18-460
Out: March 12, 2025
Due: March 27, 2025 by 11:59 pm ET

Instructions

- **Collaboration Policy:** You may collaborate with other students, but you MUST credit the students with whom you worked. You may NOT get solutions from external sources. This includes previous solution sets, solutions from students who took related courses in the past, and proofs you find online. If you get stuck, you may work with your classmates and/or come to office hours for a hint.
- **Submission Instructions:** Assignments should be submitted as pdfs on Gradescope. Each problem should be on a separate page. When submitting to Gradscope, make sure to label each question using the interface provided by Gradescope.
- **Programming:** Programming assignments should also be submitted to Gradescope. If you are not planning to use MATLAB or Python for your submission, please clear your choice with the instructors before submitting.

Problem 1: Stochastic Gradient Descent (25 pts)

You will use stochastic gradient descent to train a multi-class Support Vector Machine (SVM) with hinge loss. The training dataset is $\mathbf{S} = \{(x_i, y_i)\}_{i=1}^m$, where $x_i \in \mathbb{R}^{p+1}$, $y_i \in \{1, \cdots, C\}$, m is the number of samples, C is the number of classes, p is the number of features (each x_i is $p+1$-dimensional, the first p dimention is the feature, and the last dimension is fixed to be 1).

The SVM model contains weights $w^{(1)}$, $w^{(2)}$, ..., $w^{(C)}$ with each $w^{(j)} \in \mathbb{R}^{p+1}$ being a weight vector for class j. Given the weights and an input x, the SVM model predicts the class as $\hat{y} = \arg\max_{j \in \{1,\cdots,C\}} (w^{(j)})^\top x$.

To train the SVM model, we minimize the following objective function:

$$
\begin{aligned}
f(W) &= \frac{1}{2} \sum_{j=1}^{C} \|w^{(j)}\|_2^2 + \frac{\lambda}{m} \sum_{i=1}^{m} \sum_{j=1}^{C} \left(1 - \mathbb{1}\{y_i = j\} (w^{(j)})^\top x_i\right)_+^2 \\
&= \frac{1}{m} \sum_{i=1}^{m} \underbrace{\left[\frac{1}{2} \sum_{j=1}^{C} \|w^{(j)}\|_2^2 + \lambda \sum_{j=1}^{C} \left(1 - \mathbb{1}\{y_i = j\} (w^{(j)})^\top x_i\right)_+^2 \right]}_{f_i(W)} \\
&= \frac{1}{m} \sum_{i=1}^{m} f_i(W)
\end{aligned}
$$

where

$$
\begin{aligned}
W &= [w^{(1)}, \cdots, w^{(C)}] \in \mathbb{R}^{(p+1) \times C} \\
\mathbb{1}\{y_i = j\} &= \begin{cases} 1 & \text{if } y_i = j \\ -1 & \text{if } y_i \neq j \end{cases}
\end{aligned}
$$

and notation $(a)_+$ means $(a)_+ = \max\{a, 0\}$.

1. (9 pts) Gradient computation: Derive the gradient of the objective function $f(W)$ w.r.t the parameter W. You may write the gradient either in the matrix form, or separately write down the gradient for each $w^{(j)}$.

2. Implement batch gradient descent and mini-batch stochastic gradient descent algorithms in order to minimize $f(W)$. In both cases, initialize the values of W as zero, and update the parameters for 200 epochs (the meaning of epoch is explained below).

   - (8 pts) **Batch Gradient Descent**: In this setting, each epoch means doing one gradient step using the exact gradient of f. Plot the values of the objective function, train accuracy, test accuracy after each epoch for $\lambda = 0.1, 0.5, 1, 10$. Use step size as 0.005. Interpret your results.

- (8 pts) **Mini-Batch Stochastic Gradient Descent**: For each epoch, randomly shuffle the dataset and divide the dataset evenly into mini-batches, each of size 5000. Update the parameters using the average gradient of data from each minibatch (i.e. $\frac{1}{|B|}\sum_{i\in B}\nabla f_i(W)$ where B is the minibatch), and sequentially iterate through all mini-batches of the dataset. After iterating through all minibatches in this dataset, the epoch is finished and we move on to the next epoch and repeat the above. **Note that iterating through the entire data set once is counted as one epoch**. Plot the value of the objective function, train accuracy, test accuracy after each epoch for $\lambda = 0.1, 1, 30, 50$. Use step size as 0.001. Interpret your results.

Dataset description: The training and test data consists of features extracted from CIFAR-10 images. It contains the following csv files:

- *train_features.csv*: training set features (last column is all 1's)
- *train_labels.csv*: training set class labels
- *test_features.csv*: test set features (last column is all 1's)
- *test_labels.csv*: test set class labels

The training set consists of $m = 50,000$ examples and the test set consists of $10,000$ examples. Each example is in a new line and is $p+1 = 401$ dimensional. There are $C = 10$ classes in this dataset. As mentioned before, for a given weight W and input x, we predict the class as $\hat{y} = \arg\max_{j\in\{1,\cdots,C\}}(w^{(j)})^\top x$. Given W, the training accuracy is the fraction of data points predicted correctly in the training set, and the test accuracy is the fraction of data points predicted correctly in the test set.

Problem 2: Training a Neural Network with Gradient Methods (15 pts)

You will apply a few popular gradient descent algorithms to train a three-layer neural network with cross-entropy loss on the CIFAR-10 dataset (the same dataset you used in Problem 1). A starter code for this problem is provided in `p2.py`, which you should review to understand its functionality.

You will experiment with four gradient descent variants: minibatch-SGD, Adam, RMSProp, and AdaGrad. PyTorch has built-in implementations of these algorithms, and the starter code provides a simplified interface for using them. For detailed parameter descriptions, consult the PyTorch documentation. For each algorithm, run training for 50 epochs with a batch size of 128.

- (3 pts) **Mini-Batch Stochastic Gradient Descent**: Plot the value of the training loss, training accuracy, and test accuracy after each epoch with learning rate $0.001, 0.01, 0.1$. Interpret your results.

- (4 pts) **Adam**: Plot the value of the training loss, training accuracy, and test accuracy after each epoch with learning rate $0.001, 0.01, 0.1$. Now fix the learning rate to be 0.1. Plot the value of the training loss, training accuracy, test accuracy after each epoch with $\beta_1 = \beta_2 = 0$. Compare the result with the previous ones, which used the default parameters $\beta_1 = 0.9, \beta_2 = 0.999$. Interpret your results.

- (3 pts) **RMSProp**: Plot the value of the training loss, training accuracy, and test accuracy after each epoch with learning rate $0.001, 0.01, 0.1$. Interpret your results.

- (3 pts) **AdaGrad**: Plot the value of the training loss, training accuracy, and test accuracy after each epoch with learning rate $0.001, 0.01, 0.1$. Interpret your results.

- (2 pts) **Comparison:** Use the best hyperparameter setting from the previous experiments, plot the training loss, training accuracy, and test accuracy (3 plots total), where each plot has one curve for each of minibatch-SGD, Adam, RMSProp, and AdaGrad.

Problem 3: Subgradients (30 pts)

1. (12 pts) Derive the subdifferential for the following functions for all the points in their domain.

   (a) $f(x) = \max(x, x^2)$ over domain $\mathbb{R}$.

   (b) $f(x) = x_1^2 + 4x_2^2 + |x_1 - x_2|$ over domain $\mathbb{R}^2$.

   (c) $f(x) = \|Ax - b\|_2^2 + \lambda\|x\|_1$ over domain $\mathbb{R}^p$. Here $A \in \mathbb{R}^{n\times p}, b \in \mathbb{R}^n$, $\lambda > 0$.

2. (18 pts, Subgradient vs Proximal Gradient Descent) Consider the following cost function $f(x) = \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1$, where the domain is $\mathbb{R}^p$ and $A \in \mathbb{R}^{n\times p}$ and $b \in \mathbb{R}^n$.

   (a) (8 pts) Write down the update equation for subgradient method and proximal gradient method. For proximal gradient method, treat the $\frac{1}{2}\|Ax - b\|_2^2$ as the smooth part, and $\lambda\|x\|_1$ as the non-differentiable part. You need to write down the explicit formula for how to calculate the prox operator.

   (b) (10 pts) Implement subgradient method with fixed step size, subgradient method with vanishing step size and proximal gradient method using the given data. Data file `homework3_p3_data.mat` contains variables:

   - `lambda`: this is the value of λ.
   - `As`: $n \times p \times$ trials ($100 \times 500 \times 100$), `As(:,:,i)` is the predictor matrix of the ith trial
   - `bs`: $n \times$ trials (100×100), `bs(:,i)` is the response vector of the ith trial
   - `x_stars`: $p \times$ trials (500×100), `x_stars(:,i)` is the optimizer of the ith trial

   We are going to do 100 trials, each with different A, b in the cost function. In each trial i, we use `As(:,:,i)` as the A matrix, and `bs(:,i)` as the b vector, and run the respective algorithms for 10,000 iterations. You may initialize x as 0. Choose step size as 0.001 for subgradient method with fixed step size and proximal gradient method. For subgradient with vanishing step size, use $\frac{0.001}{\sqrt{k}}$ as the step size where k is the iteration counter. For each iteration k between 1 and 10,000, record the optimality gap $f(x(k)) - f^* := gap^i(k)$ for all 3 algorithms, where $f^* = f(x^*)$ and the value of the optimizer x^* is in `x_stars(:,i)` in the data file.

After you are done with all the trials, plot the average optimality gap across all trials as a function of k, i.e. plot $\frac{1}{100}\sum_{i=1}^{100} gap^i(k)$ vs. k for all three algorithms in the same figure. Use log scale for y axis.

Problem 4: Projection onto ℓ_1 ball (30 pts)

To project a point $v \in \mathbb{R}^n$ onto the unit ℓ_1-ball, we solve the following problem

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{2}\|x - v\|_2^2 \\ \text{subject to } & \|x\|_1 \leq 1. \end{aligned}$$

1. (15 pts) Write down the Lagrangian and formulate the dual problem. Explain how the optimizer of the dual problem can be used to find the optimizer of the primal problem. (Hint: you may want to use the soft thresholding function when deriving the dual function.)

2. (10 pts) Prove that the optimal primal-dual pair (x^*, λ^*) satisfies the following: if $\|v\|_1 \leq 1$, $(v, 0)$ is the optimal pair; otherwise, λ^* must satisfy

$$\sum_{i=1}^{n} (|v_i| - \lambda^*)_+ = 1,$$

   where $a_+ = \max\{0, a\}$.

3. (5 pts, **optional for 18-460 students**) Propose an efficient algorithm to find (x^*, λ^*) based on part 2 of this problem. You may assume that $|v_1| \leq |v_2| \leq \ldots \leq |v_n|$.