

# Homework 3: Convex Optimization / Machine Learning (I)

CMU 15-780: Graduate AI (Spring 2023)

Out: Mar 5, 2023  
Due: Mar 22, 2023 11:59 pm

## Instructions

### Collaboration Policy

You may discuss assignments with other students as you work through them, but writeups must be done alone. No downloading or copying of code or other answers is allowed. If you use a string of at least 5 words from some source, you must cite the source. Do not use any third-party libraries except those explicitly stated in this handout.

### Submission

You should submit two files via Gradescope: a completed `problems.py` for the programming portion, and a PDF for your answers to the written component. Your completed functions will be autograded by running through several test cases (not just the ones supplied with starter code).

You have 8 late days for homeworks over the semester, and can use at most 3 for one homework.

## 1 Written: Gradient Descent [50 pts]

Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a differentiable convex function, and consider running gradient descent on  $f$ . That is, pick some  $x_0 \in \mathbb{R}^n$ , and, for some step size  $\eta > 0$ , do

$$x_{t+1} = x_t - \eta \nabla f(x_t).$$

for all  $t \geq 0$ . Assume that  $f$  is  $L$ -smooth for some  $L > 0$ , that is,

$$f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{L}{2} \|y - x\|_2^2 \quad (1)$$

for all  $x, y \in \mathbb{R}^n$ .<sup>1</sup> In class, we showed that gradient descent in this setting converges to a global optimum:

**Theorem 1** (Proved in class). *Let  $f$  be any  $L$ -smooth, differentiable, and convex function. Then, for every  $t \geq 0$ , gradient descent on  $f$  with  $\eta = 1/L$  satisfies*

$$f(x_t) - f(x^*) \leq \frac{L}{2t} \|x_0 - x^*\|_2^2.$$

for any minimizer  $x^*$  of  $f$ .

In this problem, we will use an extra assumption about  $f$  to obtain an exponentially-faster convergence result. Let  $\mu > 0$ . We say that  $f$  is  $\mu$ -strongly convex if

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2} \|y - x\|_2^2 \quad (2)$$

for every  $x, y \in \mathbb{R}^n$ . For some intuition:

---

<sup>1</sup>For convex  $f$ , this is equivalent to  $\nabla f$  being  $L$ -Lipschitz, i.e.,  $\|\nabla f(y) - \nabla f(x)\|_2 \leq L\|y - x\|_2$

- Check for yourself that a differentiable function is 0-strongly convex if and only if it is convex.
- Compare the definitions of smoothness (1) and strong convexity (2).

Your goal in this part will be to prove the following theorem:

**Theorem 2.** *Let  $f$  be any  $L$ -smooth,  $\mu$ -strongly-convex, differentiable, and convex function, where  $0 < \mu \leq L$ . Then, for every  $t \geq 0$ , gradient descent on  $f$  with  $\eta = 1/L$  satisfies*

$$\|x_t - x^*\|_2^2 \leq \left(1 - \frac{\mu}{L}\right)^t \|x_0 - x^*\|_2^2$$

where  $x^*$  is the unique minimizer of  $f$ .

1. [10 points] Theorem 2 speaks of a unique minimizer  $x^*$  of  $f$ . Show that every  $\mu$ -strongly-convex  $f$ , where  $\mu > 0$ , indeed has a unique minimizer. That is, for any  $x, y \in \mathbb{R}^n$ , show that

$$f(x) = f(y) = \min f \quad \text{implies} \quad x = y.$$

2. [10 points] Suppose  $f$  is  $L$ -smooth. Show that

$$f(x) - \min f \geq \frac{1}{2L} \|\nabla f(x)\|_2^2$$

for every  $x \in \mathbb{R}^n$ .

(Hint: We showed a similar statement in class by plugging the gradient descent update rule, namely  $y = x - (1/L)\nabla f(x)$ , into (1). Try that.)

3. [20 points] Suppose that  $f$  satisfies the conditions in Theorem 2. Show that, with  $\eta = 1/L$ , we have

$$\|x_{t+1} - x^*\|_2^2 \leq \left(1 - \frac{\mu}{L}\right) \|x_t - x^*\|_2^2$$

for every  $t \geq 0$ .

(Hint: Your proof should apply both the previous question and the definition of  $\mu$ -strong convexity.)

4. [10 points] Using the previous result, finish the proof of Theorem 2.

## 2 Programming: Support Vector Machines [50 pts]

The SVM (Support Vector Machine) is a supervised machine learning algorithm typically used for binary classification problems. There are many libraries that allow you to implement an SVM without writing a bunch of code (e.g., scikit learn, cvxopt). To get a good grasp of the important components involved in training an SVM model, we will implement a linear soft-SVM from scratch.

Make sure you have the Python packages `numpy`, `keras`, and `tensorflow` installed by running `pip3 install numpy keras tensorflow` in your shell.

1. [6 points] **Data preprocessing.** First, we prepare our target classification problem, namely the MNIST dataset. The MNIST dataset (Modified National Institute of Standards and Technology) is a large database of handwritten digits. MNIST contains a collection of 70000  $28 \times 28$  images of handwritten digits from 0 to 9. It commonly used for training various image processing systems.

For time efficiency, we use only part of the dataset to train and test our SVM model. In this assignment you will classify a digit as either a target number or not using support vector machines. After data preprocessing, the data is composed of the digits 4 and 5. We label images of the target digit 4 positively as  $+1$  and label images of the non-target digit 5 negatively as  $-1$ .

First, we perform some basic preprocessing steps to put the data into a cleaner format.

- *Vectorization* is the process of transforming a  $28 \times 28$  image to a  $28^2$ -dimensional vector.

- *Normalization* is the process of mapping a range of values into the  $[0, 1]$  interval. Normalization improves the speed of learning (e.g. faster convergence in standard gradient descent) and prevents numerical overflow.
- *Padding* is the process of adding an extra column with all 1s to the dataset. Each  $28^2$ -dimensional vector becomes a  $(28^2 + 1)$ -dimensional vector with an additional element 1. We explain the necessity of padding in the next section.

Implement the functions `vectorize`, `normalize`, and `pad` in `problems.py`.

2. [8 points] **Loss function.** SVM finds a hyperplane that separates samples of two classes with a maximum margin. It is given by the formula  $f(x) = w^\top x + b$ ,  $f(x) \geq 1$  for positively classified examples and  $f(x) \leq -1$  for negatively classified examples. See Wikipedia ([https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)) for an illustration.

Training an SVM consists of finding the weight vector  $w^*$  and intercept  $b^*$  that define this optimal hyperplane. We do this by minimizing a loss function.

Our objective is to find a hyperplane that separates the positive and negative examples with the largest margin, while keeping the misclassification rate as low as possible. Thus, the loss function  $L$  we use is defined as

$$L(w) = \frac{\lambda}{2} \|w\|_2^2 + \frac{1}{N} \sum_{i=1}^N \max(0, 1 - y_i(w^\top x_i)),$$

where  $(x_1, y_1), \dots, (x_N, y_N)$  are the points in the training set (each  $y_i \in \{-1, 1\}$ ). The first term  $\frac{\lambda}{2} \|w\|_2^2$  is inversely proportional to the margin; thus minimizing the first term has the same effect as maximizing the margin. The second term corresponds to the misclassification loss (this is known as *hinge loss*). Finally,  $\lambda$  regulates the effects between maximizing the margin and minimizing the misclassification rate. A larger  $\lambda$  favors wider margins, and vice versa.

You may have noticed that the intercept  $b$  is missing. That's because we have pushed it into the weight vector, as follows:

$$f(x) = w^\top x + b = \tilde{w}^\top \tilde{x} \text{ where } \tilde{w} = [w, b] \text{ and } \tilde{x} = [x, 1].$$

That is why we padded the input with an extra column of 1s.

Implement the loss function  $L$  in the `get_loss` function in `problems.py`.

3. [8 points] **Loss function gradient.** Implement the gradient of the loss function  $\nabla L$  in the function `get_loss_gradient` in `problems.py`. We will use this in the training phase to minimize  $L$ .
4. [10 points] **Training an SVM using gradient descent.** We minimize the loss  $L$  using gradient descent. Since our loss function  $L$  is convex, our gradient steps should always decrease the loss function, provided the step size/learning rate is small enough. A simple check when writing these optimization procedures is to print your loss values and verify that they are decreasing.

Recall that gradient descent works as follows. We start with some initial vector of weights  $w_0 \in \mathbb{R}^{28^2+1}$ , and repeat the following for  $t = 0, \dots, E - 1$  iterations:

- Find the gradient of the loss function  $\nabla L(w)$ .
- Move in the direction opposite to the gradient with a learning rate  $\eta$ :  $w_{t+1} \leftarrow w_t - \eta \cdot \nabla L(w_t)$ .

Here  $E$  denotes the number of *epochs* of gradient descent, where in each epoch  $t = 0, \dots, E - 1$ , the algorithm scans through the entire training set of examples to compute the gradient (in gradient descent, each gradient step is computed relative to the entire training set  $(x_1, y_1), \dots, (x_N, y_N)$ ).

Implement gradient descent in the function `gradient_descent` in `problems.py`.

Debugging tips: you may want to print out the loss  $L(w_t)$  at each gradient step  $t = 0, \dots, E - 1$  and check that the loss is decreasing.

5. [10 points] **Batch stochastic gradient descent with early stopping.** In the implementation of gradient descent, each gradient step uses all the training examples  $(x_1, y_1), \dots, (x_N, y_N)$ . In applications,  $N$  can be prohibitively large, leading to costly computation per gradient step. In batch stochastic gradient descent (SGD), we compute each gradient step using only  $B$  training examples. More specifically, in each epoch, we randomly divide the  $N$  training examples into  $\lfloor N/B \rfloor$  batches of size  $B$ , and run a gradient step on each batch. So each epoch consists of  $\lfloor N/B \rfloor$  gradient steps. We repeat this process for  $E$  epochs.

*Early stopping.* If  $L(w)$  has already converged, we do not need to run additional epochs of SGD. More precisely, we stop training when the loss has not decreased passed a certain threshold  $\varepsilon$ , that is, when

$$L(w_{t+1}) > L(w_t) - \varepsilon$$

where  $w_t$  is the weight vector at the end of the  $t$ th epoch. Implement batch SGD with the early stopping criterion in the function `stochastic_gradient_descent` in `problems.py`.

6. [8 points] **Testing your trained SVM.** After training your model using gradient descent or stochastic gradient descent, we get learned weights  $\hat{w}$ . Let's measure the test accuracy of the learned weights  $\hat{w}$ .

Implement the functions `predict` and `get_accuracy` in `problems.py`.

In `problems.py` we have provided you with some hyperparameters to initialize gradient descent and stochastic gradient descent. These should yield a high test accuracy and run in less than a second. You will be able to observe the relative run-times required by both algorithms to achieve a similar level of accuracy.