

# Homework 1: Linear Regression & Gradient Descent

**Due:** Tuesday, October 1, 2024 at 12 pm EST (noon)

**Instructions:** Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. If you are using LyX, then the listings package tends to work better.

**Code Submission:** Please submit all of your code in a single zip file on Gradescope.

**Overall Submission:** Please submit both a single PDF file and a zip file containing code to Gradescope.

## 1 Polynomial regression as linear least squares

In practice,  $P_{\mathcal{X} \times \mathcal{Y}}$  is usually unknown and we use the empirical risk minimizer (ERM). We will reformulate the problem as a  $d$ -dimensional linear regression problem. First note that functions in  $\mathcal{H}_d$  are parametrized by a vector  $\mathbf{b} = [b_0, b_1, \dots, b_d]^\top$ , we will use the notation  $f_{\mathbf{b}}$ . Similarly we will note  $\mathbf{a} \in \mathbb{R}^3$  the vector parametrizing  $g(x) = f_{\mathbf{a}}(x)$ . We will also gather data points from the training sample in the following matrix and vector:

$$X = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ 1 & x_2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^d \end{bmatrix}, \quad \mathbf{y} = [y_1, y_2, \dots, y_N]^\top. \quad (1)$$

These notations allow us to take advantage of the very effective linear algebra formalism.  $X$  is called the design matrix.

1. Show that the empirical risk minimizer (ERM)  $\hat{\mathbf{b}}$  is given by the following minimization  $\hat{\mathbf{b}} = \arg \min_{\mathbf{b}} \|X\mathbf{b} - \mathbf{y}\|_2^2$ .
2. If  $N > d$  and  $X$  is full rank, show that  $\hat{\mathbf{b}} = (X^\top X)^{-1} X^\top \mathbf{y}$ . (Hint: you should take the gradients of the loss above with respect to  $\mathbf{b}$ <sup>1</sup>). Why do we need to use the conditions  $N > d$  and  $X$  full rank?

## 2 Gradient descent for ridge/linear regression

### Dataset

We have provided you with a file called `ridge_regression_dataset.csv`. Columns `x0` through `x47` correspond to the input and column `y` corresponds to the output. We are trying to fit the data using a linear model and gradient based methods. Please also check the supporting code in `skeleton_code.py`. Throughout this problem, we refer to particular blocks of code to help you step by step.

<sup>1</sup>You can check the linear algebra review here if needed <http://cs229.stanford.edu/section/cs229-linalg.pdf>

### Feature normalization

When feature values differ greatly, we can get much slower rates of convergence of gradient-based algorithms. Furthermore, when we start using regularization, features with larger values are treated as “more important”, which is not usually desired.

One common approach to feature normalization is perform an affine transformation (i.e. shift and rescale) on each feature so that all feature values in the training set are in  $[0, 1]$ . Each feature gets its own transformation. We then apply the same transformations to each feature on the validation set or test set. Importantly, the transformation is “learned” on the training set, and then applied to the test set. It is possible that some transformed test set values will lie outside the  $[0, 1]$  interval.

1. Modify function `feature_normalization` to normalize all the features to  $[0, 1]$ . Can you use numpy’s broadcasting here? Often broadcasting can help to simplify and/or speed up your code. Note that a feature with constant value cannot be normalized in this way. Your function should discard features that are constant in the training set.

At the end of the skeleton code, the function `load_data` loads, split into a training and test set, and normalize the data using your `feature_normalization`.

### Linear regression

In linear regression, we consider the hypothesis space of linear functions  $h_\theta : \mathbb{R}^d \rightarrow \mathbb{R}$ , where

$$h_\theta(x) = \theta^T x,$$

for  $\theta, x \in \mathbb{R}^d$ , and we choose  $\theta$  that minimizes the following “average square loss” objective function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}_i) - y_i)^2,$$

where  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \mathbb{R}^d \times \mathbb{R}$  is our training data.

While this formulation of linear regression is very convenient, it’s more standard to use a hypothesis space of affine functions:

$$h_{\theta,b}(x) = \theta^T x + b,$$

which allows a nonzero intercept term  $b$  – sometimes called a “bias” term. The standard way to achieve this, while still maintaining the convenience of the first representation, is to add an extra dimension to  $x$  that is always a fixed value, such as 1, and use  $\theta, x \in \mathbb{R}^{d+1}$ . Convince yourself that this is equivalent. We will assume this representation.

2. Let  $X \in \mathbb{R}^{m \times (d+1)}$  be the *design matrix*, where the  $i$ ’th row of  $X$  is  $\mathbf{x}_i$ . Let  $y = (y_1, \dots, y_m)^T \in \mathbb{R}^{m \times 1}$  be the *response*. Write the objective function  $J(\theta)$  as a matrix/vector expression, without using an explicit summation sign. <sup>2</sup>
3. Write down an expression for the gradient of  $J$  without using an explicit summation sign.
4. Write down the expression for updating  $\theta$  in the gradient descent algorithm for a step size  $\eta$ .

---

<sup>2</sup>Being able to write expressions as matrix/vector expressions without summations is crucial to making implementations that are useful in practice, since you can use numpy (or more generally, an efficient numerical linear algebra library) to implement these matrix/vector operations orders of magnitude faster than naively implementing with loops in Python.

5. Modify the function `compute_square_loss`, to compute  $J(\theta)$  for a given  $\theta$ . You might want to create a small dataset for which you can compute  $J(\theta)$  by hand, and verify that your `compute_square_loss` function returns the correct value.
6. Modify the function `compute_square_loss_gradient`, to compute  $\nabla_{\theta} J(\theta)$ . You may again want to use a small dataset to verify that your `compute_square_loss_gradient` function returns the correct value.

### Gradient checker

We can numerically check the gradient calculation. If  $J : \mathbb{R}^d \rightarrow \mathbb{R}$  is differentiable, then for any vector  $h \in \mathbb{R}^d$ , the directional derivative of  $J$  at  $\theta$  in the direction  $h$  is given by

$$\lim_{\epsilon \rightarrow 0} \frac{J(\theta + \epsilon h) - J(\theta - \epsilon h)}{2\epsilon}.$$

It is also given by the more standard definition of directional derivative,

$$\lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} [J(\theta + \epsilon h) - J(\theta)] .$$

The former form gives a better approximation to the derivative when we are using small (but not infinitesimally small)  $\epsilon$ . We can approximate this directional derivative by choosing a small value of  $\epsilon > 0$  and evaluating the quotient above. We can get an approximation to the gradient by approximating the directional derivatives in each coordinate direction and putting them together into a vector. In other words, take  $h = (1, 0, 0, \dots, 0)$  to get the first component of the gradient. Then take  $h = (0, 1, 0, \dots, 0)$  to get the second component, and so on.

7. Complete the function `grad_checker` according to the documentation of the function given in the `skeleton_code.py`. Alternatively, you may complete the function `generic_grad_checker` so which can work for any objective function.

You should be able to check that the gradients you computed above remain correct throughout the learning below.

### Batch gradient descent

We will now finish the job of running regression on the training set.

8. Complete `batch_gradient_descent`. Note the phrase *batch* gradient descent distinguishes between *stochastic* gradient descent or more generally *minibatch* gradient descent.
9. Now let's experiment with the step size. Note that if the step size is too large, gradient descent may not converge. Starting with a step-size of 0.1, try various different fixed step sizes to see which converges most quickly and/or which diverge. As a minimum, try step sizes 0.5, 0.1, .05, and .01. Plot the average square loss on the training set as a function of the number of steps for each step size. Briefly summarize your findings.
10. For the learning rate you selected above, plot the average test loss as a function of the iterations. You should observe overfitting: the test error first decreases and then increases.

### Ridge Regression

We will add  $\ell_2$  regularization to linear regression. When we have a large number of features compared to instances, regularization can help control overfitting. *Ridge regression* is linear

regression with  $\ell_2$  regularization. The regularization term is sometimes called a penalty term. The objective function for ridge regression is

$$J_\lambda(\theta) = \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}_i) - y_i)^2 + \lambda \theta^T \theta,$$

where  $\lambda$  is the regularization parameter, which controls the degree of regularization. Note that the bias term (which we included as an extra dimension in  $\theta$ ) is being regularized as well as the other parameters. Sometimes it is preferable to treat this term separately.

11. Compute the gradient of  $J_\lambda(\theta)$  and write down the expression for updating  $\theta$  in the gradient descent algorithm. (Matrix/vector expression, without explicit summation)
12. Implement `compute_regularized_square_loss_gradient`.
13. Implement `regularized_grad_descent`.

Our goal is to find  $\lambda$  that gives the minimum average square loss on the test set. So you should start your search very broadly, looking over several orders of magnitude. For example,  $\lambda \in \{10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}, 1, 10, 100\}$ . Then you can zoom in on the best range. Follow the steps below to proceed.

14. Choosing a reasonable step-size, plot training average square loss and the test average square loss (just the average square loss part, without the regularization, in each case) as a function of the training iterations for various values of  $\lambda$ . What do you notice in terms of overfitting?
15. Plot the training average square loss and the test average square loss at the end of training as a function of  $\lambda$ . You may want to have  $\log(\lambda)$  on the  $x$ -axis rather than  $\lambda$ . Which value of  $\lambda$  would you choose ?

### 3 Image classification with regularized logistic regression

#### Dataset

We will examine a classification problem. To do so we will use the MNIST dataset<sup>3</sup> which is one of the traditional image benchmark for machine learning algorithms. We will only load the data from the 0 and 1 class, and try to predict the class from the image. You will find the support code for this problem in `mnist_classification_source_code.py`. Before starting, take a little time to inspect the data. Load `X_train`, `y_train`, `X_test`, `y_test` with `pre_process_mnist_01()`. Using the function `plt.imshow` from `matplotlib` visualize some data points from `X_train` by reshaping the 764 dimensional vectors into  $28 \times 28$  arrays. Note how the class labels '0' and '1' have been encoded in `y_train`. No need to report these steps in your submission.

#### Logistic regression

We will use here again a linear model, meaning that we will fit an affine function,

$$h_{\theta,b}(\mathbf{x}) = \theta^T \mathbf{x} + b,$$

with  $\mathbf{x} \in \mathbb{R}^{764}$ ,  $\theta \in \mathbb{R}^{764}$  and  $b \in \mathbb{R}$ . This time we will use the logistic loss instead of the squared loss. Instead of coding everything from scratch, we will also use the package `scikit learn` and

<sup>3</sup><http://yann.lecun.com/exdb/mnist/>

study the effects of  $\ell_1$  regularization. You may want to check that you have a version of the package up to date (0.24.1).

16. Recall the definition of the logistic loss between target  $y$  and a prediction  $h_{\theta,b}(\mathbf{x})$  as a function of the margin  $m = yh_{\theta,b}(\mathbf{x})$ . Show that given that we chose the convention  $y_i \in \{-1, 1\}$ , our objective function over the training data  $\{\mathbf{x}_i, y_i\}_{i=1}^m$  can be re-written as

$$L(\theta) = \frac{1}{2m} \sum_{i=1}^m (1 + y_i) \log(1 + e^{-h_{\theta,b}(\mathbf{x}_i)}) + (1 - y_i) \log(1 + e^{h_{\theta,b}(\mathbf{x}_i)}).$$

17. What will become the loss function if we regularize the coefficients of  $\theta$  with an  $\ell_1$  penalty using a regularization parameter  $\alpha$ ?

We are going to use the module `SGDClassifier` from scikit learn. In the code provided we have set a little example of its usage. By checking the online documentation<sup>4</sup>, make sure you understand the meaning of all the keyword arguments that were specified. We will keep the learning rate schedule and the maximum number of iterations fixed to the given values for all the problem. Note that scikit learn is actually implementing a fancy version of SGD to deal with the  $\ell_1$  penalty which is not differentiable everywhere, but we will not enter these details here.

18. To evaluate the quality of our model we will use the classification error, which corresponds to the fraction of incorrectly labeled examples. For a given sample, the classification error is 1 if no example was labeled correctly and 0 if all examples were perfectly labeled. Using the method `clf.predict()` from the classifier write a function that takes as input an `SGDClassifier` which we will call `clf`, a design matrix `X` and a target vector `y` and returns the classification error. You should check that your function returns the same value as `1 - clf.score(X, y)`.

To speed up computations we will subsample the data. Using the function `sub_sample`, restrict `X_train` and `y_train` to `N_train = 100`.

19. Report the test classification error achieved by the logistic regression as a function of the regularization parameters  $\alpha$  (taking 10 values between  $10^{-4}$  and  $10^{-1}$ ). You should make a plot with  $\alpha$  as the x-axis in log scale. For each value of  $\alpha$ , you should repeat the experiment 10 times so has to finally report the mean value and the standard deviation. You should use `plt.errorbar` to plot the standard deviation as error bars.
20. Which source(s) of randomness are we averaging over by repeating the experiment?
21. What is the optimal value of the parameter  $\alpha$  among the values you tested?
22. Finally, for one run of the fit for each value of  $\alpha$  plot the value of the fitted  $\theta$ . You can access it via `clf.coef_`, and should reshape the 764 dimensional vector to a  $28 \times 28$  array to visualize it with `plt.imshow`. Defining `scale = np.abs(clf.coef_).max()`, you can use the following keyword arguments (`cmap=plt.cm.RdBu`, `vmax=scale`, `vmin=-scale`) which will set the colors nicely in the plot. You should also use a `plt.colorbar()` to visualize the values associated with the colors.
23. What can you note about the pattern in  $\theta$ ? What can you note about the effect of the regularization?

<sup>4</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.SGDClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html)