

ECE 581: Least Squares and Projection for MNIST

Henry D. Pfister
Duke University

November 23, 2025

1 Motivating Questions

1.1 How does one fit a polynomial to a set of points?

For a given set of N data points $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N) \in \mathbb{R}^2$, what degree- m polynomial,

$$f(x) = \sum_{j=0}^m a_j x^j,$$

best fits the data? Of course, the answer depends on what we mean by “best”. If we focus on the *mean-squared error* (MSE), then the goal is to minimize the quantity

$$\frac{1}{N} \sum_{i=1}^N (y_i - f(x_i))^2$$

over the polynomial coefficients $a_0, a_1, \dots, a_m \in \mathbb{R}$. If $N > m + 1$, then the polynomial may not fit each point exactly and the result that minimizes this error is called the *least-squares solution*.

1.2 How can one learn to classify patterns from training examples?

1.3 Binary Case

Consider a system, which can be in one of two states, that generates independent random vectors using a fixed distribution for each state. Suppose we are given N observations of the system,

$$(\underline{x}_1, y_1), (\underline{x}_2, y_2), \dots, (\underline{x}_N, y_N),$$

where $\underline{x}_i \in \mathbb{R}^n$ is the observed vector and $y_i \in \{-1, 1\}$ is the system state, and asked to classify a new observation $\underline{x} \in \mathbb{R}^n$ for which the state is unknown.

One approach to this problem is to fit a function $f: \mathbb{R}^n \rightarrow \{-1, 1\}$ to the given N observations and then use that function to classify \underline{x} . This approach is known as *empirical risk minimization*. Specifically, we define a non-negative function $L: \{-1, 1\} \times \{-1, 1\} \rightarrow \mathbb{R}_{\geq 0}$ that quantifies the loss $L(\hat{y}, y)$ incurred by classifying the true state y to the estimate \hat{y} . Then, we minimize the empirical risk

$$R_N(f) \triangleq \frac{1}{N} \sum_{i=1}^N L(f(\underline{x}_i), y_i)$$

over all functions f in a some class \mathcal{F} . The resulting function is denoted by

$$\hat{f} = \arg \min_{f \in \mathcal{F}} R_N(f).$$

To compare with the optimal function, we assume that the training data is generated by i.i.d. samples from some true distribution $P_{\underline{X}, Y}(\underline{x}, y)$. An important result from learning theory is that, if the space \mathcal{F} of functions has nice properties and N is large enough, then \hat{f} will be close to the function

$$f^* = \arg \min_{f \in \mathcal{F}} R(f) = \arg \min_{f \in \mathcal{F}} \mathbb{E}[L(f(\underline{X}), Y)]$$

that minimizes the expected risk $R(f) \triangleq \mathbb{E}[L(f(\underline{X}), Y)]$, where the expectation over $P_{\underline{X}, Y}(\underline{x}, y)$. To minimize the error, one can choose

$$L(\hat{y}, y) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{if } y = \hat{y}. \end{cases}$$

But, for computational reasons, this approach is sometimes relaxed to use other loss functions. For example, one can fit a function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ with the squared loss function

$$L(\hat{y}, y) = (y - \hat{y})^2.$$

Then, the vector is classified by rounding the output to the set $\{-1, 1\}$.

1.4 Non-Binary Case

For $d > 2$ classes, the output set of class labels naturally becomes $\{1, 2, \dots, d\}$. Likewise, one can define a loss function $L: \{1, \dots, d\} \times \{1, \dots, d\} \rightarrow \mathbb{R}_{\geq 0}$ that maps pairs of class labels to a real loss.

In practice, it more common to encode each label $y \in \{1, \dots, d\}$ into a one-hot vector $\underline{e}_y \in \{0, 1\}^d$ whose y -th coordinate is 1. In this case, the optimal fit can also be relaxed to fit a real function $f: \mathbb{R}^n \rightarrow \mathbb{R}^d$ with least-squares the hybrid loss function

$$L(f(\underline{x}), y) = \|\underline{e}_y - f(\underline{x})\|^2.$$

To get a decision, the vector \underline{x} can be classified by choosing the index of the largest element in the vector $\underline{s} = f(\underline{x})$,

$$\hat{y} = \arg \max_{i \in \{1, \dots, d\}} s_i.$$

2 Linear Least-Squares

The most common linear least-squares problem is the solution of an overdetermined system of linear equations. Let $A \in \mathbb{R}^{N \times M}$ be an $N \times M$ matrix with $N > M$ that defines N linear equations in M variables,

$$y_i = \sum_{j=1}^M A_{i,j} z_j.$$

When $N > M$, it is likely that there is no \underline{z} vector that satisfies all of these equations. Thus, one often asks for a least-squares solution that minimizes the squared error

$$\hat{\underline{z}} = \arg \min_{\underline{z}} \left(\frac{1}{N} \sum_{i=1}^N \left(y_i - \sum_{j=1}^M A_{i,j} z_j \right)^2 \right).$$

As noted in the handout on inner product spaces, if A is full rank, then this solution can be written as

$$\hat{\underline{z}} = (A^T A)^{-1} A^T \underline{y}.$$

This setup naturally extends to fitting a vector space of functions $\mathcal{F} = \text{span}\{f_1, f_2, \dots, f_M\}$, where $f_j: \mathbb{R}^n \rightarrow \mathbb{R}$ for $j = 1, \dots, M$, to a set of input/output pairs, $(\underline{x}_1, y_1), (\underline{x}_2, y_2), \dots, (\underline{x}_N, y_N)$. In particular, we define

$$f(\underline{x}; \underline{z}) \triangleq \sum_{j=1}^M z_j f_j(\underline{x}),$$

choose $L(\hat{y}, y) = (y - \hat{y})^2$, and observe that

$$\begin{aligned} R_N(f) &\triangleq \frac{1}{N} \sum_{i=1}^N L(f(\underline{x}_i; \underline{z}), y_i) \\ &= \frac{1}{N} \sum_{i=1}^N (y_i - f(\underline{x}_i; \underline{z}))^2 \\ &= \frac{1}{N} \sum_{i=1}^N \left(y_i - \sum_{j=1}^M z_j f_j(\underline{x}_i) \right)^2 \\ &= \frac{1}{N} \sum_{i=1}^N \left(y_i - \sum_{j=1}^M A_{i,j} z_j \right)^2, \end{aligned}$$

where we define $A_{i,j} \triangleq f_j(\underline{x}_i)$. Thus, using this setup, one can minimize the empirical risk by finding a least-squares solution to the linear system $\underline{y} = A\underline{z}$. For example, if A has full rank and $N > M$, then one can use the formula $\underline{z} = (A^T A)^{-1} A^T \underline{y}$.

In some cases, it is more computationally efficient (or convenient) to minimize $R_N(f)$ using gradient descent. Letting $\underline{a}_i = [A_{i,1}, \dots, A_{i,M}]$ be the i -th row of A , the gradient of the empirical risk is given by

$$\nabla_{\underline{z}} \frac{1}{N} \sum_{i=1}^N (y_i - \underline{a}_i \cdot \underline{z})^2 = \frac{1}{N} \sum_{i=1}^N \nabla_{\underline{z}} (y_i - \underline{a}_i \cdot \underline{z})^2 = -\frac{2}{N} \sum_{i=1}^N (y_i - \underline{a}_i \cdot \underline{z}) \underline{a}_i.$$

Since the gradient is the sum of N simple terms, a standard approach is to use the terms separately to update \underline{z} during each step. Starting from $\underline{z}^{(0)} = \underline{0}$, the update for cyclic partial gradient descent is

$$\underline{z}^{(t+1)} = \underline{z}^{(t)} + \gamma_t \left(y_{(t \bmod N)+1} - \underline{a}_{(t \bmod N)+1} \cdot \underline{z}^{(t)} \right) \underline{a}_{(t \bmod N)+1},$$

where γ_t is the step size.

Exercise 1. (5 pts Vandermonde function) When $n = 1$, we can fit a degree- m polynomial by choosing $f_j(x) = x^{j-1}$ and $M = m + 1$. In this case, it follows that $A_{i,j} = x_i^{j-1}$ and the matrix A is called a Vandermonde matrix. Implement a function to compute a Vandermonde matrix given the x values and polynomial degree.

Exercise 2. (10 pts solve_linear_LS function + 5 pts print MSE + 5 pts plot) Using the setup in the previous example, fit the points $(1, 2), (2, 3), (3, 5), (4, 7), (5, 11), (6, 13)$ to a degree-2 polynomial using linear algebra. What is the resulting mean squared error? Plot the resulting polynomial (for $x \in [0, 7]$) along with the data points to see the quality of fit.

Exercise 3. (10 pts solve_linear_LS_gd function + 5 pts print MSE + 5 pts plot) Using the setup in the previous problem, fit the given points to a degree-2 polynomial using cyclic partial gradient descent. For the fixed step size $\gamma_t = 0.0002$, determine a value of T such that $\underline{z}^{(T)}$ achieves a mean squared error that is at most 20% larger than the previous answer. Plot this polynomial (for $x \in [0, 7]$) along the previous polynomial and the data points.

3 Training a Linear Classifier via Least Squares

3.1 Two Classes

A linear classifier is a linear function that classifies points $\underline{x} \in \mathbb{R}^n$ into two classes by testing

$$\sum_{i=1}^n x_i z_i \gtrless 0, \tag{1}$$

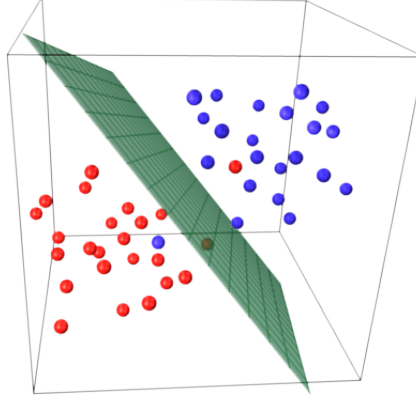


Figure 1: This hyperplane is designed to separate two classes in three dimensions. It classifies all but two points correctly.

with ties broken arbitrarily (see Figure 1). Since the LHS of (1) equals the standard inner product $\langle \underline{x} | \underline{z} \rangle$, this value is quite related to the vector projection of \underline{x} onto \underline{z} . In fact, if $\|\underline{z}\| = 1$, then the LHS of (1) equals the coefficient of \underline{z} for the vector projection of \underline{x} onto \underline{z} . The orthogonal complement of \underline{z} is the hyperplane whose normal vector is \underline{z} and the LHS of (1) equals zero for all points in that hyperplane. Therefore, that hyperplane is the *decision boundary* for the test in (1).

Mathematically, the space \mathbb{R}^n is divided into two disjoint sets by a hyperplane containing the origin. To allow instead for a general hyperplane, which may not contain the origin, one can append a “−1” to the end of each \underline{x} data vector (i.e., extending the length by one). In that case, we may assume that $x_n = -1$ and this formula can be rewritten as

$$\sum_{i=1}^{n-1} x_i z_i \geq z_n,$$

which defines a general hyperplane separation for $\underline{x} \in \mathbb{R}^{n-1}$.

One can train a linear classifier using least-squares by performing empirical risk minimization with the loss function $L(\hat{y}, y) = (y - \hat{y})^2$ and function class

$$\mathcal{F} = \left\{ f(\underline{x}) = \sum_{i=1}^n x_i z_i \mid z_1, \dots, z_n \in \mathbb{R} \right\}.$$

This problem can be solved using the approach described above with $f_j(\underline{x}) = [\underline{x}]_j = x_j$, which implies that $A_{i,j} = [\underline{x}_i]_j$. To evaluate a trained classifier, one needs additional samples that are independent from the samples used in training. In practice, this can be achieved by breaking the original data set into two pieces: a training set and a test set. This is called *cross validation*¹.

For the trained classifier, one should report the *classification error rate* (i.e., the fraction of data set that is misclassified) for both the training and test sets. Also, a more detailed way to present classification error rates is with a confusion matrix. A *confusion matrix* C is a matrix that contains whose $C_{i,j}$ entry equals the number of times a data vector is predicted to be in class j (by the classifier) when the its true class is i .

3.2 Many Classes

For $d > 2$ classes, encode each label $y \in \{1, \dots, d\}$ as a one-hot vector $\underline{e}_y \in \{0, 1\}^d$ whose y -th coordinate is 1. Stack the features into a matrix $X \in \mathbb{R}^{N \times n}$ and the one-hot labels into $Y \in \mathbb{R}^{N \times d}$. To handle bias, we define X_b as the new input data where either $X_b = X$ and $n' = n$ or we append a bias term

¹More generally, the set can be broken into k pieces and one can use k -fold cross validation.

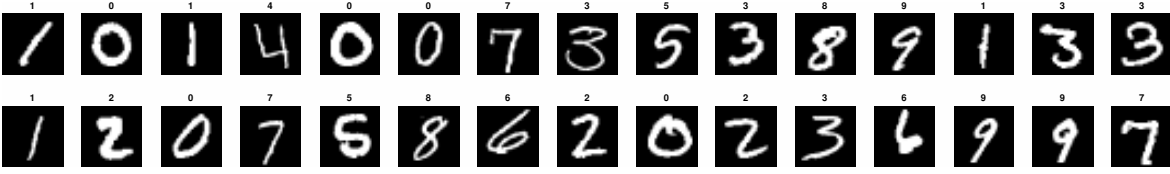


Figure 2: Sample digits from the MNIST database.

which gives $X_b = [X \ 1]$ and set $n' = n + 1$. A linear multi-output model learns a matrix $W \in \mathbb{R}^{n' \times d}$ that minimizes the following regularized empirical risk

$$\min_W \|X_b W - Y\|_F^2 + \lambda \|W\|_F^2,$$

whose solution is

$$\widehat{W} = (X_b^\top X_b + \lambda I)^{-1} X_b^\top Y.$$

Given a new input \underline{x} , we form x_b where we append a 1 if required. Then, we can compute scores $\hat{\underline{s}} = \widehat{W}^\top x_b \in \mathbb{R}^d$ and choose the predicted class to be the index of the largest real value:

$$\hat{y} = \arg \max_{i \in \{1, \dots, d\}} \hat{s}_i.$$

We note that this least-squares problem is equivalent to fitting d real-valued functions independently (one per column of Y).

4 Project

4.1 Dataset

Definition 1. The *MNIST database* is a set of digitized handwritten digits that is used for testing and training classifiers (see Figure 2). Each sample in the dataset is stored as a 28×28 pixel image with each pixel taking on 256 grayscale levels along with a label (0-9). The PyTorch datasets class is the easiest way to download and the following Python snippet loads and plots the first 30 images with their labels.

```
import numpy as np
import torch
from torchvision import datasets, transforms
import matplotlib.pyplot as plt

# Load
transform = transforms.Compose([transforms.ToTensor()])
train_data = datasets.MNIST(root="./data", train=True,
                             download=True, transform=transform)
test_data = datasets.MNIST(root="./data", train=False,
                             download=True, transform=transform)

# Extract and normalize as torch tensors
X_train = (train_data.data.float().view(-1, 28*28) / 255.0).numpy()
y_train = (train_data.targets.numpy())

# Plot MNIST
plt.figure(figsize=(15, 2.5))
for i in range(30):
    x, y = X_train[i], y_train[i]
```

```
plt.subplot(2, 15, i + 1)
plt.imshow(x.reshape(28, 28), cmap='gray')
plt.axis('off')
plt.title(int(y))
```

4.2 Overview and Goals

This project mirrors the accompanying Jupyter notebook `erm_project.ipynb` and has two parts:

- Part A (Exercises 1–3): one-dimensional polynomial fitting by least squares. You will build a Vandermonde matrix, solve the normal equations to fit a degree-2 polynomial to a small dataset (report MSE and plot), and reproduce the fit using cyclic gradient descent (compare MSE and curves).
- Part B (Exercises 4–5): a linear multi-class classifier for MNIST trained by least squares with one-hot labels. First train on raw pixels and report training/testing error and a confusion matrix; then build simple local quadratic features (five features per 2×2 patch) and repeat. Optionally include a small ridge parameter $\lambda \geq 0$.

You will fill the marked cells in the notebook and may rely on the linear-algebra notes (inner products, projection theorem, normal equations) as needed.

Data. Use torchvision to download the MNIST dataset.

Packages. Python 3 with: numpy, torch, torchvision, scikit-learn, matplotlib.

4.3 Project Tasks and What to Submit

Here are the directions.

- Tasks
 - From “`erm_code.py`”, complete “Add code” lines and save as “`erm_code_soln.py`”,
 - Test your “`erm_code_soln.py`” by “`erm_code_test.py`” and reading the output,
 - Complete Python notebook “`erm_project.ipynb`”.
- Submit the executed python notebook (as ipynb and printed to pdf) with additional notes:
 - Comparing of normal equations vs. gradient descent,
 - Discussing local quadratic features and notes on the effect of regularization if used.

5 Exercises

After completing and transforming the “`erm_code.py`” file into the “`erm_code_soln.py`” file, all work is done in the notebook. Here is a summary of the exercises.

5.1 Exercise 1: Vandermonde Matrix (5 pts)

For $n = 1$, fitting a degree- m polynomial uses $A_{i,j} = x_i^{j-1}$ for $j = 1, \dots, m+1$. Implement a function that returns the $N \times (m+1)$ Vandermonde matrix for inputs x_1, \dots, x_N .

5.2 Exercise 2: Least-Squares Fit and Plot (15 pts)

Using the points $(1, 2), (2, 3), (3, 5), (4, 7), (5, 11), (6, 13)$ and degree $m = 2$, solve the least-squares problem, report the mean-squared error, and plot the fitted polynomial on $x \in [0, 7]$ along with the data.

5.3 Exercise 3: Gradient-Descent Least Squares (15 pts)

Implement cyclic partial gradient descent for the same setup. With a fixed step size, select T such that the mean-squared error is within 20% of Exercise 2. Plot both polynomials and the data.

5.4 Exercise 4: Multi-Class MNIST via Least Squares (20 pts)

Let $X \in \mathbb{R}^{N \times d}$ be features and $Y \in \mathbb{R}^{N \times 10}$ be one-hot labels. Train with the normal equations

$$\widehat{W} = (X^\top X + \lambda I)^{-1} X^\top Y,$$

predict by arg max over the 10 real outputs, and report training/testing error and a confusion matrix.

5.5 Exercise 5: Local Quadratic Features (20 pts)

For each interior pixel (i, j) , add five features from a 2×2 neighborhood: $x_{i,j}$, $x_{i,j}^2$, $x_{i,j}x_{i+1,j}$, $x_{i,j}x_{i,j+1}$, and $x_{i,j}x_{i+1,j+1}$. Re-train the least-squares classifier (as in Exercise 4) and report training/testing error and a confusion matrix. Briefly compare with raw-pixel features and note any effect of λ if used.

6 Grading Rubric (100 pts)

- Exercise 1 (Vandermonde): 5
- Exercise 2 (LS + MSE + plot): 15
- Exercise 3 (GD LS + MSE + plot): 15
- Exercise 4 (MNIST LS one-hot + errors + confusion): 20
- Exercise 5 (local quadratic features + report + comparison): 20
- Code quality and clarity; discussions and comparisons: 25