CSCI 145 Problem Set 6

October 2, 2025

Submission Instructions

Please upload *your* work by **11:59pm Monday October 6, 2025.**

- You are encouraged to discuss ideas and work with your classmates. However, you **must acknowledge** your collaborators at the top of each solution on which you collaborated with others and you **must write** your solutions independently.
- Your solutions to theory questions must be written legibly, or typeset in LaTeX or markdown. If you would like to use LaTeX, you can import the source of this document (available from the course webpage) to Overleaf.
- I recommend that you write your solutions to coding questions in a Jupyter notebook using Google Colab.
- You should submit your solutions as a **single PDF** via the assignment on Gradescope.

Grading: The point of the problem set is for *you* to learn. To this end, I hope to disincentivize the use of LLMs by **not** grading your work for correctness. Instead, you will grade your own work by comparing it to my solutions. This self-grade is due the Friday *after* the problem set is due, also on Gradescope.

Problem 1: MNIST

We will train two kinds of logistic regression models on the MNIST dataset. The first is standard logistic regression, the second will use the reparameterization trick.

Tip: When in doubt, print out the shapes of the data and the models you're working with! You can do this for a matrix `X` by calling `X.shape`.

Hint: There are several concepts I'm expecting you to figure out on your own. If you're stuck with a bug or a step, use generative AI and/or throw a message on discord.

Part A: Loading MNIST

The MNIST dataset consists of tens of thousands of images of handwritten digits and their labels (0, 1, 2, ..., 9). Load the MNIST dataset using `torch`. Define a training set of 1000 randomly selected image/label pairs, and a testing set of 100 (different) randomly selected image/label pairs. Create a dataloader for each set.

Part B: Logistic Regression

In this part, you'll find a solution to the problem:

$$\min_{\mathbf{W}\in\mathbb{R}^{d\times k}} \mathcal{L}_{\text{CrossEntropy}}(\mathbf{XW}, \mathbf{y}) \tag{1}$$

where k is the number of classes.

Using our three step recipe for machine learning, initialize a logistic regression model for the multi-class classification problem. As described in the class notes, the idea is to have one output for each possible class. Instead of the binary cross entropy loss function we used during our first foray with `torch`, we'll need the cross entropy loss this time (but you can still pass the unnormalized logits to this function).

Once you've trained your model for 100 steps, plot 10 randomly selected images from the test set and the label your model predicts. (Of the ten values for each input, the prediction your model makes is the one with the largest logit value.)

Part C: Reparameterization Trick

In this part, you'll find a solution to the problem:

$$\min_{\mathbf{V}\in\mathbb{R}^{n\times k}} \mathcal{L}_{\text{CrossEntropy}}(\mathbf{XX}^\top\mathbf{V}, \mathbf{y}). \tag{2}$$

Let's do logistic regression with the reparameterization trick. First, build your matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, and compute the Gaussian kernel matrix $\mathbf{XX}^\top = \mathbf{K} \in \mathbb{R}^{n\times n}$.

Recall:

$$[\mathbf{K}]_{i,j} = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|_2^2}{2\sigma^2}\right). \tag{3}$$

What should the size of the model input be? Train the model, but notice we can't use batches of data as we normally do.

For randomly selected test images, get predictions from your model. (You'll first need to compute the Gaussian kernel between the test points and the original data.)

Plot the images and the predicted labels.

Part D: Summary

Which variant of logistic regression train more quickly? Which scales better as we increase the number of training points?

Problem 2: Logistic Regression to Neural Network

Part A: Sigmoid Derivative

At the heart of backpropagation is the idea of modularly computing the derivative. For a function f with input z, we can compute the derivative of the loss with respect to z by adding, for all functions f that take z as input,

$$\frac{\partial \mathcal{L}}{\partial z} += \frac{\partial \mathcal{L}}{\partial f}\frac{\partial f}{\partial z}. \tag{4}$$

For functions like $f(z) = z^2$, computing $\frac{\partial f}{\partial z}$ is quite easy. In this problem, we will compute the derivative of the sigmoid function

$$\sigma(z) = \frac{1}{1+e^{-z}}. \tag{5}$$

In particular, show that

$$\frac{\partial}{\partial z}\sigma(z) = \sigma(z)(1-\sigma(z)). \tag{6}$$

Hint: Try differentiating $\log(\sigma(z))$, and solve for $\frac{\partial \sigma(z)}{\partial z}$.

Part B: Neural Network

One of the benefits of `torch` is that backpropagation is hidden under the hood. Given a number of layers ℓ, build a model

$$f^{(\ell)}(\mathbf{x}) = \mathbf{W}^{(\ell)}(\text{ReLU}(\mathbf{W}^{(\ell-1)}(\dots(\text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}))) \tag{7}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{m\times d}$, $\mathbf{W}^{(\ell)} \in \mathbb{R}^{k\times m}$, and $\mathbf{W}^{(j)} \in \mathbb{R}^{m\times m}$ for some number of neurons m. Note: When $\ell = 1$, there is only one matrix with dimension $k \times d$.

Part C: Empirical Comparison

Train your model for $\ell = 1, 2, 3$, and record the training and test loss for each epoch. Plot the training loss (solid line) and test loss (dotted line) for each depth of model. What do you notice?