EE/CS 228 HW#2 - Neural Network for MNIST

Due: Friday Jan 30th 11:59 pm

This exercise will focus on training a neural network classifier for the MNIST dataset.
Background: MNIST is a standard digit classification dataset. Given a 28×28 image containing a digit from 0 to 9, our goal is deducing which digit the image corresponds to. The dataset has 50,000 training and 10,000 test examples. The data can be downloaded from `https://github.com/cvdfoundation/mnist?tab=readme-ov-file`.

Formatting the data: Your goal will be building a binary classifier for MNIST. This classifier will

- output 1 if the input image is a digit greater than 4 (i.e. digits 5,6,7,8,9).
- output 0 if the input image is a digit less than or equal to 4 (i.e. digits 0,1,2,3,4).

Towards this goal, we need to format the data to obtain a dataset $\mathcal{S} = (\boldsymbol{x}_i, y_i)_{i=1}^{N=50{,}000}$.

- **Input:** Each input $\boldsymbol{x}$ is a 28×28 matrix. Apply the following operations to obtain $d = 785$ dimensional input features.
  - Convert inputs $\boldsymbol{x}$ to vectors of size $28^2 = 784$.
  - Standardize input images using z-normalization: Scale each image i ($1 \le i \le N$) to have zero-mean and unit variance. In Python terms, this corresponds to the operation $\boldsymbol{x} \to \frac{\boldsymbol{x} - \text{np.mean}(\boldsymbol{x})}{\text{np.std}(\boldsymbol{x})}$ where $\boldsymbol{x} \in \mathbb{R}^{784}$ is the feature vector of the i'th image. Note that mean and std are calculated across examples, and the standardization operation (subtraction and division) is entry-wise. Use the same normalization (i.e., mean and std estimated from the training data) during test.
  - Add bias variable by concatenating 1 to your input i.e. the input becomes $\boldsymbol{x} \to \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$. The input dimension becomes $d = 785$.
- **Output:** Each label y is a digit from 0 to 9. Convert y to $0, 1$ as follows:

$$y \to \begin{cases} 0 & \text{if} \quad 0 \le y \le 4 \\ 1 & \text{if} \quad 5 \le y \le 9 \end{cases}$$

Shallow Neural Net Classifier: Our goal is learning a neural net to predict if an image is greater than 4. We will use a neural net for this purpose with the following form

$$f(\boldsymbol{x}) = \boldsymbol{v}^T \text{ReLU}(\boldsymbol{W}\boldsymbol{x}) \in \mathbb{R}.$$

The decision of the neural network is then given by the hard-thresholding $\hat{y} = 1_{f(x) > 0.5}$. Here $\boldsymbol{W} \in \mathbb{R}^{k \times d}$ is the input layer and $\boldsymbol{v} \in \mathbb{R}^k$ is the output layer. We will use ReLU activation.
Optimizer: You are expected to use stochastic gradient descent with batch size 10. To keep things simple, you can use a constant learning rate. You are supposed to tune your learning rate. You should do 10 passes (each pass is called an epoch) over the data i.e. use $(N/10) \times 10 = 50{,}000$ mini-batch iterations[1].

[1] If training takes too long, you can use only N=10,000 MNIST samples and do 50,000 iterations. This will cost you 2 pts in your assignment out of 20 pts.

Initialization: It is critical to initialize the neural net properly. In this assignment, initialize your weight matrices v, W with independent and identical Gaussian distributed entries using Xavier initialization: $\boldsymbol{W}_0 \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \frac{1}{d})$ and $\boldsymbol{v}_0 \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \frac{1}{k})$.

Assignment

Your tasks are as follows. All algorithms should be your own code. No Tensorflow/Pytorch.

1. (2 pts) Apply the normalization on the training and test data.
2. (2 pts) As a baseline, train a linear classifier $\hat{y} = \boldsymbol{v}^T \boldsymbol{x}$ and quadratic loss. Report its test accuracy.
3. (7 pts) Train a neural network classifier with quadratic loss $\ell(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$. Plot the progress of the test and training accuracy (y-axis) as a function of the iteration counter t (x-axis)[2]. Report the final test accuracy for the following choices
   - k=5
   - k=50
   - k=200
   - Comment on the role of hidden units k on the ease of optimization and accuracy.
4. (7 pts) Train a neural network classifier with logistic loss, namely $\ell(y, f(\boldsymbol{x})) = -y \log(\sigma(f(\boldsymbol{x}))) - (1 - y) \log(1 - \sigma(f(\boldsymbol{x})))$ where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function. In this case, the hard-thresholding is applied on top of the sigmoid function, i.e., $\hat{y} = 1_{\sigma(f(x)) > 0.5}$. Repeat step 3.
5. (2 pts) Comment on the difference between linear model and neural net. Comment on the differences between logistic and quadratic loss in terms of optimization and test/train accuracy.

[2]In the figure, you can report the performance at every 100 or 1000 iterations if the resolution is too dense