Homework 2

Deadline: Wednesday, Oct. 14, at 11:59pm.

Submission: You need to submit the following files through MarkUs[1]:

- Your answers to Questions 1, 2, and 3, and outputs requested for Question 2 and 3, as a PDF file titled `hw2_writeup.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.
- Python files `run_knn.py`, `logistic.py`, and `run_logistic_regression.py` completed for Question 2.
- Python file `nn.py` completed for Question 3.

Neatness Point: One point will be given for neatness. You will receive this point as long as we don't have a hard time reading your solutions or understanding the structure of your code.

Late Submission: 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.

Computing: To install Python and required libraries, see the instructions on the course web page.

Homeworks are individual work. See the Course Information handout[2] for detailed policies.

1. [**4pts**] **Feature Maps.** Suppose we have the following 1-D dataset for binary classification:

x	t
-1	1
1	0
3	1

   (a) [**2pts**] Argue briefly (at most a few sentences) that this dataset is not linearly separable. (Your argument should resemble the one we used in lecture to prove XOR is not linearly separable.)

   (b) [**2pts**] Now suppose we apply the feature map

$$\boldsymbol{\psi}(x) = \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \end{pmatrix} = \begin{pmatrix} x \\ x^2 \end{pmatrix}.$$

   Assume we have no bias term, so that the parameters are w_1 and w_2. Write down the constraint on w_1 and w_2 corresponding to each training example, and then find a pair of values (w_1, w_2) that correctly classify all the examples. Remember that there is no bias term.

2. [**22pts**] **kNN vs. Logistic Regression.** In this problem, you will compare the performance and characteristics of different classifiers, namely k-Nearest Neighbors and Logistic Regression. You will complete the provided code in `q2/` and experiment with the completed code. You should understand the code instead of using it as a black box.

[1] https://markus.teach.cs.toronto.edu/csc311-2020-09
[2] http://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/syllabus.pdf

The data you will be working with is a subset of MNIST hand-written digits, 4s and 9s, represented as 28×28 pixel arrays. We show the example digits in figure 1. There are two training sets: `mnist_train`, which contains 80 examples of each class, and `mnist_train_small`, which contains 5 examples of each class. There is also a validation set `mnist_valid` that you should use for model selection, and a test set `mnist_test` that you should use for reporting the final performance. Optionally, the code for visualizing the datasets is located at `plot_digits.py`.

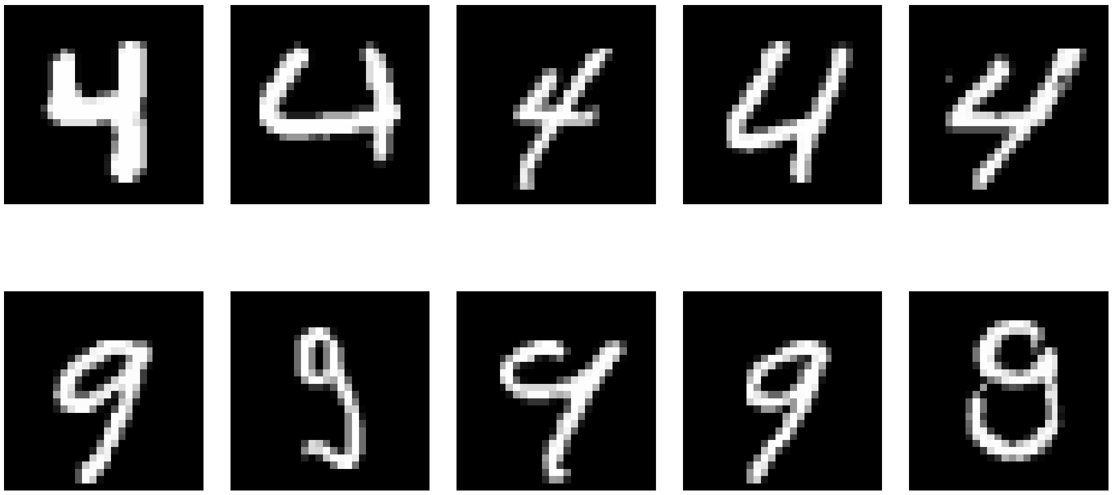

Figure 1: Example digits. Top and bottom show digits of 4s and 9s, respectively.

2.1. ***k*-Nearest Neighbors.** Use the supplied kNN implementation to predict labels for `mnist_valid`, using the training set `mnist_train`.

(a) **[2pts]** Implement a function `run_knn` in `run_knn.py` that runs kNN for different values of $k \in \{1, 3, 5, 7, 9\}$ and plots the classification rate on the validation set (number of correctly predicted cases, divided by total number of data points) as a function of k. Report the plot in the write-up.

(b) **[2pts]** Comment on the performance of the classifier and argue which value of k you would choose. What is the classification rate for k^*, your chosen value of k? Also report the classification rate for $k^* + 2$ and $k^* - 2$. How does the test performance of these values of k correspond to the validation performance[3]?

2.2. **Logistic Regression.** Read the provided code in `run_logistic_regression.py` and `logistic.py`. You need to implement the logistic regression model, where the cost is defined as:

$$\mathcal{J} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\text{CE}}(y^{(i)}, t^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \Big(-t^{(i)} \log y^{(i)} - (1 - t^{(i)}) \log(1 - y^{(i)}) \Big),$$

where N is the total number of data points.

(a) **[4pts]** Implement functions `logistic_predict`, `evaluate`, and `logistic` located at `logistic.py`.

[3]In general, you shouldn't peek at the test set multiple times, but we do this for this question as an illustrative exercise.

(b) **[5pts]** Complete the missing parts in a function `run_logistic_regression` located at `run_logistic_regression.py`. You may use the implemented functions from part (a). Run the code on both `mnist_train` and `mnist_train_small`. Check whether the value returned by `run_check_grad` is small to make sure your implementation in part (a) is correct. Experiment with the hyperparameters for the learning rate, the number of iterations (if you have a smaller learning rate, your model will take longer to converge), and the way in which you initialize the weights. If you get `NaN/Inf` errors, you may try to reduce your learning rate or initialize with smaller weights. For each dataset, report which hyperparameter settings you found worked the best and the final cross entropy and classification error on the training, validation, and test sets. Note that you should only compute the test error once you have selected your best hyperparameter settings using the validation set.

(c) **[2pts]** Examine how the cross entropy changes as training progresses. Generate and report 2 plots, one for each of `mnist_train` and `mnist_train_small`. In each plot, you need show two curves: one for the training set and one for the validation set. Run your code several times and observe if the results change. If they do, how would you choose the best parameter settings?

2.3. **Penalized logistic regression.** Next, you need to implement the penalized logistic regression model, where the cost is defined as:

$$\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_{\mathrm{CE}}(y^{(i)}, t^{(i)}) + \frac{\lambda}{2}\|\mathbf{w}\|^2.$$

Note that you should only penalize the weights and not the bias term.

(a) **[2pts]** Implement a function `logistic_pen` in `logistic.py` that computes the penalized logistic regression.

(b) **[3pts]** Complete the missing parts in a function `run_pen_logistic_regression` located at `run_logistic_regression.py`. Choose a hyperparameter setting which seems to work well (for learning rate, number of iterations, and weight initialization). With these hyperparameters, the function evaluates different values of $\lambda \in \{0, 0.001, 0.01, 0.1, 1.0\}$ automatically and re-runs (penalized) logistic regression 5 times for each value of λ. So you will have two nested loops: the outer loop is over values of λ and the inner loops is over multiple re-runs. Your code should average the training metrics (cross entropy and classification error) over the different re-runs. Train on both `mnist_train` and `mnist_train_small`, and report averaged cross entropy and classification error on the training and validation sets for each λ. Also, for each λ, select one run, and report 2 plots that shows how the cross entropy changes as training progresses, one for each of `mnist_train` and `mnist_train_small`. In each plot, you need to show two curves: one for the training set and one for the validation set. In total, you will have to generate 10 plots.

(c) **[2pts]** For each dataset, how does the cross entropy change when you increase λ? Do they go up, down, first up and then down, or down and then up? Explain why you think they behave this way. Which is the best value of λ, based on your experiments? Report the test cross entropy and classification rate for the best value of λ.

3. **[15pts] Neural Networks.** In this problem, you will experiment on a subset of the Toronto

Faces Dataset (TFD). You will complete the provided code in `q3/` and experiment with the completed code. You should understand the code instead of using it as a black box.

We subsample 3374, 419 and 385 grayscale images from TFD as the training, validation and testing set, respectively. Each image is of size 48×48 and contains a face that has been extracted from a variety of sources. The faces have been rotated, scaled and aligned to make the task easier. The faces have been labeled by experts and research assistants based on their expression. These expressions fall into one of seven categories: 1-Anger, 2-Disgust, 3-Fear, 4-Happy, 5-Sad, 6-Surprise, 7-Neutral. We show one example face per class in Figure 2.

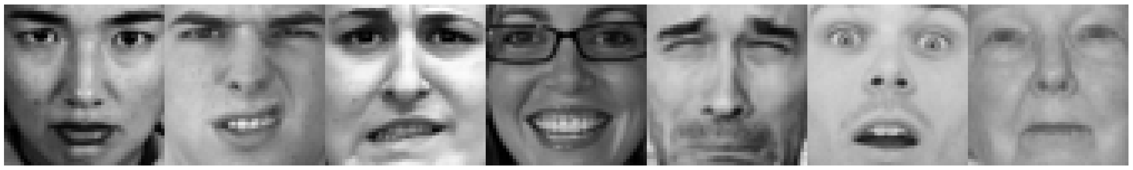

Figure 2: Example faces. From left to right, the the corresponding class is from 1 to 7.

The code for training a neural network (Multilayer Perceptrons) is partially provided in `nn.py`.

(a) **[4 pts]** Follow the instructions in `nn.py` to implement the missing functions that perform the backward pass of the network.

(b) **[2 pts]** Train the neural network with the default set of hyperparameters. Report training, validation, and testing errors and a plot of error curves (training and validation). Examine the statistics and plots of training error and validation error (generalization). How does the network's performance differ on the training set vs. the validation set during learning?

(c) **[3 pts]** Try different values of the learning rate α. Try 5 different settings from 0.001 to 1.0. What happens to the convergence properties of the algorithm (looking at both cross-entropy and percent-correct)? Try 5 different mini-batch sizes, from 10 to 1000. How does mini-batch size affect convergence? How would you choose the best value of these parameters? In each of these hold the other parameters constant while you vary the one you are studying.

(d) **[3 pts]** Try 3 different values of the number of hidden units for each layer of the Multilayer Perceptron (range from 2 to 100). You might need to adjust the learning rate and the number of epochs. Comment on the effect of this modification on the convergence properties, and the generalization of the network.

(e) **[3 pts]** Plot some examples where the neural network is not confident of the classification output (the top score is below some threshold), and comment on them. Will the classifier be correct if it outputs the top scoring class anyways?