HOMEWORK 2 - V2

CSC311 Fall 2019

University of Toronto

Version history:	V0 → V1:	remove conv-net from Q3.2
		change σ_i to σ_i^2 in Q1.1
	V1 → V2:	fix question statement in Q1.1
		clarify the prior on bias in Q1.3

- **Deadline:** Oct. 17, at 23:59.
- **Submission:** You need to submit one pdf file through MarkUs including all your answers, plots, and your code. You can produce the file however you like (e.g. LaTeX, Microsoft Word, etc), as long as it is readable. Points will be deducted if we have a hard time reading your solutions or understanding the structure of your code. For each question, you should append your code right after your answers to that question.

1. Logistic Regression - 30 pts. In this question, we will look at logistic regression from a probabilistic perspective.

1.1. *Bayes' Rule - 10 pts.* Suppose you have a D-dimensional data vector $\mathbf{x} = (x_1, \dots, x_D)^T$ and an associated class variable $t \in \{0, 1\}$ which is Bernoulli random variable with parameter α (i.e. $\mathbb{P}(t = 1) = \alpha$ and $\mathbb{P}(t = 0) = 1 - \alpha$). Assume that the dimensions of $\mathbf{x}$ are conditionally independent given t, and that the conditional distribution of each x_i is Gaussian with μ_{i0} and μ_{i1} as the means of the two classes and σ_i as their shared standard deviation, i.e. $x_i|t \sim \mathcal{N}(\mu_{it}, \sigma_i^2)$.

Use Bayes' rule to show that $p(t = 1|\mathbf{x})$ takes the form of a logistic function:

$$p(t = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + \exp\Big(-\sum_{i=1}^{D} w_i x_i - b\Big)}$$

Derive expressions for the weights $\mathbf{w} = (w_1, \dots, w_D)^T$ and the bias b in terms of the parameters of the class likelihoods and priors (i.e., μ_{i0}, μ_{i1}, σ_i and α).

1.2. *Maximum Likelihood Estimation - 10 pts.* Now suppose you are given a training set $\mathcal{D} = \{(\mathbf{x}^{(1)}, t^{(1)}), \dots, (\mathbf{x}^{(N)}, t^{(N)})\}$. Consider a binary logistic regression classifier of the same form as before:

$$p(t^{(n)} = 1|\mathbf{x}^{(n)}, \mathbf{w}, b) = \sigma(\mathbf{w}^T\mathbf{x}^{(n)} + b) = \frac{1}{1 + \exp\Big(-\sum_{i=1}^{D} w_i x_i^{(n)} - b\Big)}$$

Derive an expression for $L(\mathbf{w}, b)$, the negative log-likelihood of $t^{(1)}, \dots, t^{(n)}$ given $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ and the model parameters, under the i.i.d. assumption. Then derive expressions for the derivatives of L with respect to each of the model parameters.

1.3. *L2 Regularization - 10 pts.* Now, we treat $\mathbf{x}^{(i)}$'s as deterministic and assume that a Gaussian prior is placed on each element of $\mathbf{w}$ such that $p(w_i) = \mathcal{N}(w_i|0, 1/\lambda)$, and an "improper" flat prior on b such that $p(b) = 1$[1]. Derive an expression that is proportional to $p(\mathbf{w}, b|\mathcal{D})$, the posterior distribution of $\mathbf{w}$ and b, based on this prior and the likelihood defined above. The expression you derive must contain all terms that depend on $\mathbf{w}$ and b.

The posterior distribution for $\mathbf{w}$ and b is proportional to the product of this prior and the likelihood of $t^{(1)}, \ldots, t^{(n)}$:

$$p(\mathbf{w}, b|t^{(1)}, \ldots, t^{(n)}) \propto p(\mathbf{w})p(b)p(t^{(1)}, \ldots, t^{(n)}|\mathbf{w}, b)$$

Define $L_{\text{post}}(\mathbf{w}, b)$ to be the negative logarithm of this posterior. Show that $L_{\text{post}}(\mathbf{w}, b)$ takes the following form:

$$L_{\text{post}}(\mathbf{w}, b) = L(\mathbf{w}, b) + \frac{\lambda}{2}\sum_{i=1}^{D} w_i^2 + C$$

where C is a term that depends on λ but not on either $\mathbf{w}$ or b. What are the derivatives of L_{post} with respect to each of the model parameters?

2. Logistic Regression vs. KNN - 30 pts. In this section you will compare the performance and characteristics of different classifiers, namely Logistic Regression and k-Nearest Neighbors. You will extend the provided code and experiment with these extensions. Note that you should understand the code first instead of using it as a black box.

Python: you should use Python with both the Numpy and Matplotlib packages installed.

The data you will be working with are hand-written digits, 4s and 9s, represented as 28x28 pixel arrays. 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`.

Code for visualizing the datasets has been included in `plot_digits`.

2.1. *k-Nearest Neighbors - 5 pts.* Use the supplied kNN implementation to predict labels for `mnist_valid`, using `mnist_train` as the training set.

Write a script 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.

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 compute the rate for $k^* + 2$ and $k^* - 2$. Does the test performance for these values of k correspond to the validation performance?[2] Why or why not?

[1]This is a like a uniform distribution on the entire real line, but it is improper and doesn't really qualify for being a density.

[2]In general you shouldn't peek at the test set multiple times, but for the purposes of this question it can be an illustrative exercise.

2.2. *Logistic regression - 10 pts.* Look through the code in `logistic_regression_template` and `logistic`. Complete the implementation of logistic regression by providing the missing part of `logistic`. Use `checkgrad` to make sure that your gradients are correct.

Run the code on both `mnist_train` and `mnist_train_small`. You will need to 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.

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.

Next look at how the cross entropy changes as training progresses. Submit 2 plots, one for each of `mnist_train` and `mnist_train_small`. In each plot 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 - 15 pts.* Look through the code in `logistic_regression_template` and `logistic`. Complete the implementation of logistic regression by providing the missing part of `logistic`.

Now, implement the penalized logistic regression model you derived in Question 1.3 by modifying `logistic` to include a regularizer. Call the new function `logistic_pen`. You should only penalize the weights and not the bias term, as it only controls the height of the function but not its complexity. Note that you can omit the $C(\lambda)$ term in your error computation, since its derivative is 0 w.r.t. the weights and bias.

Run the code on both `mnist_train` and `mnist_train_small`. You will need to 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.

Choose a hyperparameter setting which seems to work well (for learning rate, number of iterations, and weight initialization). With these hyperparameters, do the following for each value of the penalty parameter $\lambda \in \{0, 0.001, 0.01, 0.1, 1.0\}$:

- Train on both `mnist_train` and `mnist_train_small`, and report cross entropy and classification error on the training and validation sets.
- Look at how the cross entropy changes as training progresses. Submit 2 plots, one for each of `mnist_train` and `mnist_train_small`. In each plot show two curves one for the training set and one for the validation set.

To do the comparison systematically, you should write a script that includes a loop to evaluate different values of λ automatically. You should also re-run logistic regression at least 5 times for each value of λ for the classification error and cross entropy reporting.

So you will need two nested loops: The outer loop is over values of λ; the inner loop is over multiple re-runs. Average the evaluation metrics (cross entropy and clasification error) over the different re-runs. In the end, plot the average cross entropy and classification error against λ; you only need to show plots for one run at each λ value. So for each of `mnist_train` and `mnist_train_small` you will have 2 plots, one plot for cross entropy and another plot for classification error. Each plot will have two curves - one for training and one for validation.

How do the cross entropy and classification error 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 error for the best value of λ.

Compare the results with and without penalty. Which one performed better for which data set? Why do you think this is the case?

3. Neural Networks (40 points). Here you will experiment on a subset of the Toronto Faces Dataset (TFD). Some code that partially implements a regular neural network is included with this assignment (in Python).

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 1.

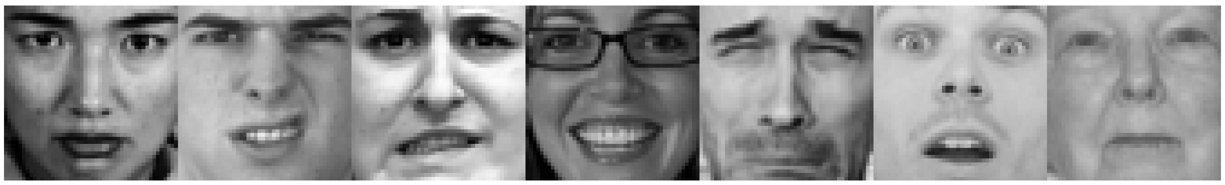

Fig 1: Example faces. From left to right, the the corresponding class is from 1 to 7.

Code for training a neural network (fully connected) is partially provided in the following files.

- `nn.py` : Train a fully connected neural network with two hidden layers.

You need to fill in some portion of the code for:

- Performing backward pass of the network.
- Performing weight update with momentum (covered in tutorial).

First, follow the instruction in the files to complete the code.

3.1. *Basic generalization [10 points].* Train a regular NN with the default set of hyperparameters. Examine the statistics and plots of training error and validation error (generalization). How does the network's performance differ on the training set versus the validation set during learning? Show a plot of error curves (training and validation) for both networks.

3.2. *Optimization [10 points].* Try different values of the learning rate ϵ ("eps"). Try 5 different settings of ϵ from 0.001 to 1.0. What happens to the convergence properties of the algorithm (looking at both cross-entropy and percent-correct)? Try 3 values of momentum from 0.0 to 0.9. How does momentum affect convergence rate? Try 5 different mini-batch sizes, from 1 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.

3.3. *Model architecture [10 points].* Fix momentum to be 0.9. Try 3 different values of the number of hidden units for each layer of the fully connected network (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.

3.4. *Network Uncertainty [10 points].* 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?