CSCE 636: Deep Learning (Fall 2022)
Assignment #1

Due 11:59PM on 9/15/2022

1. You need to submit (1) a report in PDF and (2) your code files, both to Canvas.
2. Your PDF report should include (1) answers to the non-programming part, and (2) results and analysis of the programming part. For the programming part, your PDF report should at least include the results you obtained, for example the accuracy, training curves, parameters, etc. You should also analyze your results as needed.
3. Please name your PDF report "HW#_FirstName_LastName.pdf". Please put all code files into a compressed file named "HW#_FirstName_LastName.zip". Please submit two files (.pdf and .zip) to Canvas (i.e., do not include the PDF file into the ZIP file).
4. Only write your code between the following lines. Do not modify other parts.

   ### YOUR CODE HERE

   ### END YOUR CODE
5. All students are highly encouraged to typeset their reports using Word or LaTeX. In case you decide to hand-write, please make sure your answers are clearly readable in scanned PDF.
6. Unlimited number of submissions are allowed and the latest one will be timed and graded.
7. Please read and follow submission instructions. No exception will be made to accommodate incorrectly submitted files/reports.
8. Please start your submission to Canvas at least 15-30 minutes before the deadline, as there might be latency. We do NOT accept E-mail submissions.

Linear Models for Handwritten Digits Classification: In this assignment, you will implement the binary logistic regression model and multi-class logistic regression model on a partial dataset from MNIST. In this classification task, the model will take a 16×16 image of handwritten digits as inputs and classify the image into different classes. For the binary case, the classes are 1 and 2 while for the multi-class case, the classes are 0, 1, and 2. The "data" fold contains the dataset which has already been split into a training set and a testing set. All data examples are saved in dictionary-like objects using "npz" file. For each data sample, the dictionary key 'x' indicates its raw features, which are represented by a 256-dimensional vector where the values between $[-1, 1]$ indicate grayscale pixel values for a 16×16 image. In addition, the key 'y' is the label for a data example, which can be 0, 1, or 2. The "code" fold provides the starting code. You must implement the models using the starting code.

1. Data Preprocessing [15 points]: In this problem, you need to finish "code/DataReader.py".
   (a) Explain what the function *train_valid_split* does and why we need this step.
   (b) Before testing, is it correct to re-train the model on the whole training set? Explain your answer.

(c) In this assignment, we use two hand-crafted features:
The first feature is a measure of symmetry. For a 16×16 image x, it is defined as

$$F_{symmetry} = -\frac{\sum_{pixel} |x - flip(x)|}{256},$$

where 256 is the number of pixels and $flip(\cdot)$ means left and right flipping.
The second feature is a measure of intensity. For a 16×16 image x, it is defined as

$$F_{intensity} = \frac{\sum_{pixel} x}{256},$$

which is simply the average of pixel values.
Implement them in the function *prepare_X*.

(d) In the function *prepare_X*, there is a third feature which is always 1. Explain why we need it.

(e) The function *prepare_y* is already finished. Note that the returned indices stores the indices for data from class 1 and 2. Only use these two classes for binary classification and convert the labels to +1 and -1 if necessary.

(f) Test your code in "code/main.py" and visualize the training data from class 1 and 2 by implementing the function *visualize_features*. The visualization should not include the third feature. Therefore it is a 2-D scatter plot. Include the figure in your submission.

2. Cross-entropy loss [20 points]: In logistic regression, we use the cross-entropy loss.

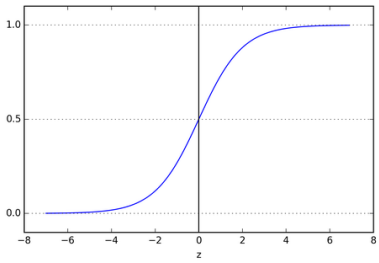
(a) Write the loss function $E(w)$ for one training data sample (x, y). Note that the binary labels are 1 and -1.

(b) Compute the gradient $\nabla E(w)$. Please provide intermediate steps of derivation.

(c) Once the optimal w is obtained, it can be used to make predictions as follows:

$$\text{Predicted class of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.5 \\ -1 & \text{if } \theta(w^T x) < 0.5 \end{cases}$$

where the function $\theta(z) = \frac{1}{1+e^{-z}}$ looks like



However, this is not the most efficient way since the decision boundary is linear. Why? Explain it. When will we need to use the sigmoid function in prediction?

(d) Is the decision boundary still linear if the prediction rule is changed to the following? Justify briefly.

$$\text{Predicted label of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.9 \\ -1 & \text{if } \theta(w^T x) < 0.9 \end{cases}$$

(e) In light of your answers to the above two questions, what is the essential property of logistic regression that results in the linear decision boundary?

3. Sigmoid logistic regression [25 points]: In this problem, you need to finish "code/LogisticRegression.py". **Please follow the instructions in the starting code. Please use data from class 1 and 2 for the binary classification.**

   (a) Based on (b) in the last problem, implement the function *_gradient*.

   (b) There are different ways to train a logistic regression model. In this assignment, you need to implement batch gradient descent, stochastic gradient descent and mini-batch gradient descent in the functions fit_BGD, fit_SGD and $fit\_miniBGD$, respectively. Note that: In batch gradient descent, each model update is based on all training samples. In stochastic gradient descent, each model update is based on one training sample. In mini-batch gradient descent, you split your training data into (roughly equal-sized) mini-batches, and each model update is based on training samples in each mini-batch. Thus, BGD and SDG are actually special cases of mini-BGD. An epoch is completed when each training sample is used to update the model exactly once. In SDG and mini-BGD, you might want to reshuffle your samples before each epoch.

   (c) Implement the functions *predict* and *score* for prediction and evaluation, respectively. Additionally, please implement the function *predict_proba* which outputs the probabilities of both classes.

   (d) Test your code in "code/main.py" and visualize the results after training by using the function *visualize_results*. Include the figure in your submission.

   (e) Implement the testing process and report the test accuracy of your best logistic regression model.

4. Softmax logistic regression [20 points]: In this problem, you need to finish "code/LRM.py". **Please follow the instructions in the starting code.**

   (a) Based on the course notes, implement the function *_gradient*.

   (b) In this assignment, you only need to implement mini-batch gradient descent in the function $fit\_miniBGD$.

   (c) Implement the functions *predict* and *score* for prediction and evaluation, respectively.

   (d) Test your code in "code/main.py" and visualize the results after training by using the function *visualize_results_multi*. Include the figure in your submission.

   (e) Implement the testing process and report the test accuracy of your best logistic regression model.

5. Softmax logistic vs Sigmoid logistic [20 points]: In this problem, you need to experimentally compare these two methods. **Please follow the instructions in the starting code.** Use data examples from class 1 and 2 for classification.

   (a) Train the softmax logistic classifier and the sigmoid logistic classifier using the same data until convergence. Compare these two classifiers and report your observations and insights.

   (b) Explore the training of these two classifiers and monitor the graidents/weights. How can we set the learning rates so that $w_1 - w_2 = w$ holds for all training steps?