CSE 326/426 (Fall 2020) Project 3

Due on 11:55pm, Nov 10, 2021

Goal: Implement an MLP for hand-written digit recognition.

Instruction:

- Read the lecture note on neural network, focusing on activation functions and the forward and backward propagation algorithms. You should use the pseudo-codes for the mini-batch stochastic gradient descent algorithm as a reference to implement the training algorithm.
- Questions in the homework related to neural networks will be useful.
- Now start coding. Decompress the file project_3.zip and find the following file structure

```
.
|-- data
|   |-- X_test.pkl
|   |-- X_train.pkl
|   |-- Y_test.pkl
|   `-- Y_train.pkl
`-- src
    |-- Visualization.ipynb
    |-- problem1.py
    |-- problem2.py
    |-- problem3.py
    |-- problem4.py
    |-- test1.py
    `-- test2.py
```

- **Dataset**: the files under the "data" folder contains a train-test split of the MNIST dataset for training and testing. For example, X_train.pkl is the vectors of training examples and Y_train.pkl contains the one-hot vectors of the labels of training examples. More information about this dataset and the current state-of-the-art classification performance can be found in https://www.kaggle.com/c/digit-recognizer/discussion/61480. The training data has 60,000 images, each of which is a square matrix with 28×28 pixels, and the test data has 10,000 images of the same format as the training images. Each pixel represents a gray level of a particular location of an image. We have reshaped the images into column vectors of length 28×28, which will be the dimension of the input layer of your neural network. Each image is a hand-written digit in $\{0, 1, \ldots, 9\}$ and the neural network needs to classify each image into one of the 10 classes. As the digits are handwritten by different persons, there are variations in the looks of the same digit.
- – problem1.py: various activation functions, including Softmax, ReLU, sigmoid, and tanh, and the cross-entropy loss function. For functions marked with "DISREGARD THIS FUNCTION" or raising the "NotImplementedError", you can safely ignore them. You should vectorize these functions as much as possible to gain speed.

- problem2.py: functions in the NN class, including forward, backward, train, and explain. The function "explain" is required for graduates only. You should vectorize these functions as much as possible to gain speed.
- problem3.py: training of your NN on the MNIST dataset. We will use this file for grading so nothing should be changed in this file.
- problem4.py: explaining the output of the neurons at the last layer. Required only for graduates. We will use this file for grading so nothing should be changed in this file.

More detailed instructions are given in the comments of the functions.

- Files test1.py and test2.py will unit-test the correctness of your implementations in the corresponding problem1.py and problem2.py files. For example, after you implement problem1.py file, run

  ```
  nosetests -v test1
  ```

  to test the correctness of your implementation of problem1.py. Note that passing the tests does not mean your implementations are entirely correct: the test can catch only a limited number of mistakes.

  If you use Anaconda (highly recommended), there should NOT be any packages you need to install. Otherwise, you will need to install nosestest for such unit test.

- We provide simple visualization of the images and the feature map to be generated by problem4.py.

  ```
  src/Visualization.ipynb
  ```

  You don't need to plot the training/test loss during training, as they are output to the terminal when we run problem3.py during grading.

Grading: This project has 100/90 points in total for graduates/undergraduates. The graduates need to implement the explain function in problem2.py to gain the extra 10 points. The number of points for each functions in problem1.py and problem2.py are printed below:

```
(5 points) Sigmoid:activate() ... ok
(5 points) Sigmoid:gradient() ... ok
(5 points) Softmax:activate() ... ok
(5 points) Tanh:activate() ... ok
(5 points) Tanh:gradient() ... ok
(5 points) ReLU:activate() ... ok
(5 points) ReLU:gradient() ... ok
(5 points) CrossEntropyLoss:loss() ... ok
(5 points) CrossEntropyLoss:gradient() ... ok
(10 points) NN:forward() ... ok
(10 points) NN:bacward() ... ok
(5 points) NN:update_parameters() ... ok
(10 points) NN:explain() Required for graduate ... ok

----------------------------------------------------------------------
Ran 13 tests in 0.863s

OK
```

20 points go to the training quality, such as **speed** (should finish running 200 passes of the training images within 3 minutes on a 2015 MacBook. We will cutoff training after 4 minutes and use whatever test accuracy your can achieve at the end – make sure your test accuracy is convergent rather than divergent), **accuracy** (I have above 96% test accuracy after the 120 passes), when running problem3.py.

The output of running problem3.py should look similar to the following:

```
num_classes = 10
num_pixels = 784
num_training_samples = 60000
===========MNIST===========
, test error = 0.16369999999999996
, test error = 0.12339999999999995
, test error = 0.10419999999999996
, test error = 0.09150000000000003
, test error = 0.08350000000000002
, test error = 0.07640000000000002
, test error = 0.07110000000000005
, test error = 0.06559999999999999
, test error = 0.06240000000000001
, test error = 0.057699999999999974
, test error = 0.054200000000000026
, test error = 0.050799999999999956
, test error = 0.04920000000000002
, test error = 0.04730000000000001
, test error = 0.045599999999999974
, test error = 0.04400000000000004
, test error = 0.04279999999999995
, test error = 0.04179999999999995
, test error = 0.040100000000000025
, test error = 0.03820000000000001
Classification report for the neural network
              precision    recall  f1-score   support

           0      0.967     0.988     0.977       980
           1      0.982     0.985     0.984      1135
           2      0.957     0.954     0.956      1032
           3      0.952     0.959     0.956      1010
           4      0.960     0.965     0.963       982
           5      0.961     0.946     0.954       892
           6      0.967     0.966     0.966       958
           7      0.965     0.954     0.959      1028
           8      0.948     0.958     0.953       974
           9      0.955     0.938     0.946      1009

   micro avg      0.962     0.962     0.962     10000
   macro avg      0.961     0.961     0.961     10000
weighted avg      0.962     0.962     0.962     10000
Confusion matrix
[[ 968    0    1    1    0    3    3    1    2    1]
 [   0 1118    4    1    0    0    4    1    7    0]
 [   6    2  985    9    4    3    4    7   10    2]
 [   1    0    7  969    0   10    1    8   11    3]
 [   1    0    6    0  948    0    4    4    3   16]
 [   6    1    3   16    0  844   10    2    7    3]
 [  10    3    4    1    6    6  925    1    2    0]
 [   1    9   13    2    4    0    0  981    2   16]
 [   3    0    5    9    4    6    5    5  933    4]
 [   5    5    1   10   21    6    1    7    7  946]]
```

The project total counts towards 10% of the final grade (40% for all projects).

Submitting: There is no hand-written report required, and your submission should include the ONLY file

```
<your_LIN>_P3.zip
```

which, when decompressed, contains **ONLY problem1.py and problem2.py** with your implementations. Please make sure you exclude the data folder since the input and output files are quite large and will delay the grading. We will take 5 point off for including those large data files. Submit the compressed file to Coursesite.

For graduates only: we grade the "explain" using test2.py. The plots in Visualization.ipynb are just for your reference. The example output figures within the plotting notebook should look similar to this:

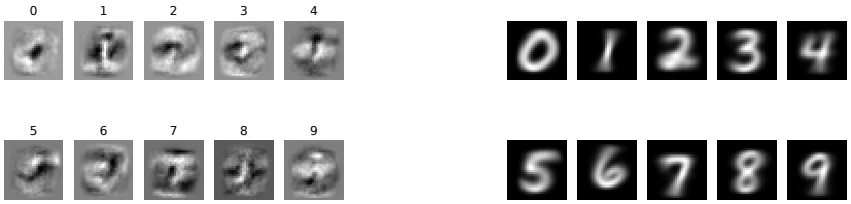

Figure 1:

Figure 2: *Left*: the feature maps generated by the explain function for each of the 10 output neurons (before the softmax activation). *Right*: average of the images from each of the 10 classes.