EECE 5136/6036: Intelligent Systems

Homework 3: Given 10/14/20; Due 11/3/20

The two homeworks so far have involved very small problems. This homework will focus on a problem closer to real-world application size, though still quite small compared to full-fledged applications. It will require complex programming, simulation and reporting, and will take a *long* time. That's why you are being given almost 3 weeks to do it. You will need them, so please start immediately.

This and Homework 4 (also being posted at the same time) are, in fact, two parts of a single large homework. This one is due Nov 3, and is the one that involves almost all the programming work. Homework 4 will require very little additional coding, and will mainly use the programs you develop for Homework 3 on the same dataset. You should keep this in mind when writing the program for this homework.

You will use the same training and testing sets for all problems in this homework and the next, so do the split once and save the sets at the beginning.

It is extremely important that you save your final networks obtained in both the problems in Homework 3 (i.e., their final weight matrices), as well as the error on every test data point, the learning rate and momentum used.

1. (300 points) One of the most widely used data sets for evaluating machine learning applications in the image analysis area is the MNIST dataset (http://yann.lecun.com/exdb/mnist/), which provides images of handwritten digits and letters. In this homework, you will use the numbers subset from this dataset. (If you are interested in the larger dataset with handwritten characters, look at https://www.nist.gov/itl/products-and-services/emnist-dataset).

Two data files are included:

- **Image Data File:** *MNISTnumImages5000.txt* is a text file that has data for 5,000 digits, each a grayscale image of size 28 $\times$ 28 pixels (i.e., 784 pixels each). Each row of the data file has 784 values representing the intensities of the image for one digit between 0 and 9. The first hundred images are shown in the included file *first100.jpg.*
- **Label Data File:** *MNISTnumLabels5000.txt* is a text file with one integer in each row, indicating the correct label of the image in the corresponding row in the image data file. Thus, the first entry '7' indicates that the first row of the image data file has data for a handwritten number 7.

You need to do the following:

1. Write a program implementing multi-layer feed-forward neural networks and training them with back-propagation including momentum. Your program must be able to handle *any* number of hidden layers and hidden neurons, and should allow the user to specify these at run-time.
2. Randomly choose 4,000 data points from the data files to form a training set, and use the remaining 1,000 data points to form a test set. Make sure each digit has equal number of points in each set (i.e., the training set should have 400 0s, 400 1s, 400 2s, etc., and the test set should have 100 0s, 100 1s, 100 2s, etc.) *You will use the same training and testing sets for all problems in this homework and the next, so make sure you save these.*
3. Train a 1-hidden layer neural network to recognize the digits using the training set. You will probably need a fairly large number of hidden neurons – in the range of 100 to 200 – but keep it below 300. You should use 10 output neurons – one for each digit – such that the correct neuron is required to produce a 1 and the rest 0. To evaluate performance during training, however, you can use the J_2 loss function with "threshold values" such as 0.75 for 1 and 0.25 for 0, as discussed in class, i.e., you will use 1 and 0 as target values but once the output becomes higher than 0.75 for a target of 1 and lower than 0.25

for a target of 0, you will not change weights in that step. To calculate performance, you can use the output neuron with the highest output as 1 and the rest as 0.

You will probably need hundreds of epochs for learning, so consider using stochastic gradient descent, where only a random subset of the 4,000 points is shown to the network in each epoch. The performance of the network in any epoch is measured by the balanced accuracy of classified points in that epoch. Save this value at the beginning, and then in every tenth epoch. Since your training and test sets are balanced (equal number of points from each class), the balanced accuracy is simply the fraction of correctly classified points (accuracy – also called hit-rate).

4. After the network is trained, test it on the test set. To evaluate performance on the test data, you can use a *winner-take-all* approach, where you consider the output correct if the correct output neuron produces the largest output among all 10 output neurons.

Write a report providing the following information. Each item required below should be placed in a separate section with the heading given at the beginning of the item.:

- **System Description:** A description of all the choices you made – number of hidden neurons, learning rate, momentum, output thresholds, rule for choosing initial weights, criterion for deciding when to stop training, etc. You may need to experiment with several parameter settings and hidden-layer sizes before you get good results.
- **Results:** Report performance of the final network on the training set and the test set using a *confusion matrix*. This is a 10×10 matrix with one row and one column for each of the classes (digits). In the (i, j) cell of the matrix, you will put the number of class i items classified as class j. Thus, cell (2, 4) of the confusion matrix will show how many 2s were (incorrectly) classified as 4. Of course, the diagonal will indicate the correct classifications. You will get one confusion matrix for the training set and another for the test set.

  Also plot the time series of the error (1 - balanced accuracy) during training using the data saved at every tenth epoch.
- **Analysis of Results:** You should describe, discuss and interpret the results you got, and why you think they are as they are.

The text part of the report, excluding the program but including the figures, should be no more than 3 pages, 12 point type, single spaced.

2. (200 points) In this problem you will train an *auto-encoder network* using the same data as in Problem 1, i.e., the same training and test sets. In this case, the goal is not to classify the images, but to obtain a good set of features for representing them. Using the simulator developed in Problem 1, you will set up a one hidden-layer feed-forward network with 784 inputs, 784 output neurons and *the same number of hidden neurons as your final network in Problem 1.* The input presented to the network will be one 28 × 28 image at a time, and the goal of the network will be to produce exactly the same image at the output. Thus, the network is learning a *reconstruction task* rather than a classification task.

The network will be trained using back-propagation with momentum. Since this is not a classification problem, and the goal is to produce real-valued outputs, you will use the J_2 loss function to quantify error, as we used when deriving back-propagation in class. The J_2 loss is given by:

$$J_2^q = \frac{1}{2}\sum_{i=1}^{784}(x_i^q - \hat{y}_i^q)^2 \tag{1}$$

where q indexes the data point, x_i^q is the value of the ith pixel in the input image, and $\hat{y}_i^q$ is the output of the ith output neuron for this data point. If the loss values are too large (because you are adding 784 squared errors), you can divide by 784, but look at this carefully because the squared error value for each pixel will typically be small. If you do divide by 784, make sure to do this also in calculating the deltas for the output layer in back-propagation.

The system will be trained on the same training set of 4,000 data points and tested on the other 1,000 points as used in Problem 1. During training, you will calculate the value of the loss function at the beginning and in every tenth epoch, and save this. Training should continue until the loss function on the training set is sufficiently low. At the end, you should also calculate the loss function over the test set. No confusion matrices are calculated because they apply only to classification problems.

After training is complete, each hidden neuron can be seen as having become tuned to a particular 28 × 28 feature for which it produces the strongest output. To visualize the feature for any hidden neuron, take its 784 weights and plot them as a 28 × 28 grayscale image (the first 28 numbers are the first row, the next 28 the second, and so on). This image will show what input the hidden neuron is most responsive for. You may need to normalize the weights between 0 and 1 or 0 and 255 before showing the image for the feature.

Write a report providing the following information. Each item required below should be placed in a separate section with the heading given at the beginning of the item.:

- **System Description:** A description of all the parameter choices you made – learning rate, momentum, rule for choosing initial weights, criterion for deciding when to stop training, etc. Again, you may need to try several parameter values. However, the number of hidden neurons in this case should be the same as the final network in Problem 1.

- **Results:** Report the performance of the final network on the training set and the test set using the loss function. In this case, this will just be two values, which you should plot as two bars side by side. Then plot the same error in the same way, but separated by each digit – so there will be two bars for 0, two for 1, etc. Also plot the time series of the error during training using the data saved at every tenth epoch.

- **Features:** Randomly select 20 hidden neurons and plot the images for their features, just like the data is shown in *first100.jpg.* Also plot the feature images for the corresponding 20 neurons in the hidden layer from Problem 1. Each set of images should be plotted as a 4×5 set of square images, with both sets shown on the same page so they can be compared. Comment on any differences or similarities that you see. What did you expect?

- **Sample Outputs:** Choose 8 samples randomly from your test set. For each of these show (one above the other) the original (input) image and the output image produced by the network after training.

Thus, you will have two rows of 8 images each. The upper row will be the actual images, and the lower row will be the output images for the same inputs.

- **Analysis of Results:** Describe, discuss and interpret the results you got, and why you think they are as they are. In particular, comment on the features you found, and what they suggest. Also comment on which digits turned out to be easiest to reconstruct and which ones more difficult, and why you think that happened.

The text part of the report, including the figures, should be no more than 4 pages, 12 point type, single spaced.

Note on Figures:
Each figure in each answer should be given a distinct name (Figure 1.2, Figure 2.1, etc.) and caption, and should be referred to by its name in the text – not as "the figure below" or "the next figure". If you have multiple panels in one figure, they should be labeled (a), (b), (c), etc., and referred to in the text as Figure 2.3(a), Figure 3.1(c), etc., and not as "the top left panel of Figure 2.3". Refer to the sample report from Homework 1 for style in general.

Appendix: Programs: The text of the complete code for each of the problems should be given as an appendix. Note that the core program used in Problem 2 is the backpropagation code written in Problem 1, so you can simply call it as a function in the code for Problem 2, which will be quite short – mainly having to do with keeping track of data and results. You may use any programming language, but you *cannot* use toolboxes, libraries, or other simulators that provide pre-programmed versions of backpropagation, feed-forward networks, or autoencoders. *You must write full programs yourself for each case.*

As in previous homeworks, the report should not be mixed in with the program. It should be a stand-alone document with text, tables, figures, etc., with the program as an appendix. *None of the information required in the report should be given as a comment or note in the program. It must all be in the report.*

Report Instructions:

Please follow the instructions given for previous homeworks.

Submission Instructions:

You should submit your report on-line through Canvas. Your submission will have three documents: 1) A report as described above; 2) A file (or .zip file) with all your source code; and 3) A brief README file with instructions on how to compile (if needed) and run the code. If Canvas does not let you submit a file with the type postfix (.py, .m, etc.) of a source code file, try changing the postfix manually to .docx or .txt and submit it. State in the README file what the postfix should be changed to in order to run the code.

Grading:

Points will be awarded for correctness of the results, proper plots, and the clarity of description and conclusions.

You may consult your colleagues for ideas, but *please write your own programs and your own text.* Both the text of the reports and the code will be checked randomly for similarity with that of other students. *Any text or code found to be copied will lead to an automatic zero on the entire homework for all students involved.* Repeat offenses in the future will incur more severe consequences.

If you cannot access the data, please send me mail at. Ali.Minai@uc.edu.