IE 7615 Problem Set

Note: The number of this Problem-Set is the assignment number as listed on Canvas.

SUBMISSION INSTRUCTIONS:

Due-date: The due-date for this homework assignment is as listed on Canvas (see "Due" within the Assignment page). The completed assignment needs to be submitted into Canvas by that due-date.

What to submit:

1. Self-grading file **selfgrades_X.txt**, where X is the Canvas assignment number of this homework. This file is provided and it **must** be used for this homework. Complete (i.e., fill in) the file exactly according to the instructions provided below, and include it with your submission. Do not use any other self-grading file with this homework, and do not combine the self-grading file with any other file(s) you submit as part of this homework.

2. Solutions to programming problems.

   You need to submit all of the code, and the products of its execution, including the results and plots. If your code is correct, the results of its execution should look similar (and the resulting numbers approximately equal) to the ones appearing within the respective sections of this homework assignment.

   Notes for Jupyter users: If you use Jupyter, you may submit your results in the form of the print-out from your Jupyter notebook execution. This print-out needs to be in a pdf file format, and needs to include all of the code and the results of its execution.
   Do not submit your code files as .ipynb files. You **must** convert your Jupyter Notebook .ipynb files to plain-text (.txt) files prior to submission, and submit your codes as .txt files.

   For any ".py" file you submit, you **must** change its file-extension to ".txt" prior to submission. Any file formats other than ".pdf" or ".txt" are **not** allowed.

Do **not** zip, tar, or compress in any other way the files you submit. Any files you submit must be uncompressed. Any files you submit must be either in pdf (i.e., ".pdf') or text (i.e., ".txt") formats. Any file formats other than ".pdf' or ".txt" are **not** allowed.

Reminder: Copying solutions from **any** source – e.g., online, solutions manuals, assignment solutions from prior semesters, or solutions from any other sources – is prohibited.

SELF-GRADING (self-assessment):

To provide you an opportunity to practice self-assessing your own work, you must self-grade the results of your work and you must submit these self-grades along with your homework and exactly in the manner defined in these instructions. We will randomly choose a subset of submissions to verify the grades.

How to self-grade problems: The number of points you are allowed to claim for any particular problem is listed in the problem section-headings.

Instructions for completing self-grading files:

To receive any credit for this assignment, a separate self-grading text file **must** be included as part of this assignment submission. Use the self-grading text file that is provided with this assignment. You need to edit that file by filling in your self-assigned grades, and submit the edited file with your self-grades filled in. The submitted self-grading file must be **exactly** in format and according to the following specifications.

Self-grading file must contain **one line only** (i.e., single line inside the file). This line must contain text fields separated by be **commas**, i.e., comma-separated format. The first three comma-separated

fields must contain (in this order) your last name, your first name, and your Northeastern University student email address, respectively. The remaining fields must contain your self-assigned grades for **all** gradable problems (in the order in which those problems appear in the assignment).

Gradable problem is any problem in this assignment for which a point-value (maximum number of points you can obtain for that particular problem) has been listed in the heading of that problem's respective section or subsection (look for square parentheses, e.g., [X points]).

The last field of the self-grading file line must contain your TOTAL self-grade for this homework assignment. Obviously, this total must be equal to the sum of all problem self-grades.

Do the following to complete your self-grading file:

1. Get the "starter" self-grading file, provided within this assignment on Canvas. This file contains the following single line:

   Lastname,Firstname,Email,

2. Replace last-name, first-name, and email-address fields with your data. You are reminded that the email address must be your Northeastern University email address.

3. Continuing this same single line, enter your self-grades for all gradable problems, in the order of gradable problems and including any zero-grades you may have self-assigned for any of the problems.

   Each self-assigned grade must be followed by a **single comma** (i.e., self-grade fields are comma-separated fields).

   Make sure that the number of these comma-separated self-grade fields equals the number of gradable problems. Do not skip self-grades of any gradable problems. If you have not attempted a particular problem, enter self-grade of (numerical zero) for that problem. Do **not** use any blanks.

4. Put a comma after the last self-grade field and then continue in the same line by entering the total self-grade for this homework. Do not add any characters after the total self-grade field. Other than filling the fields in the single line according to the specifications provided here, do **not** alter anything in the file format. Do **not** add any extra lines.

   Example:

   - Johnny Doe is listed in Northeastern University student records as: LAST=Doe, FIRST=Charles, MIDDLE=John.
   - Suppose a hypothetical homework assignment consisting of six gradable problems (six here is just an example) with their maximum point-values of 15, 15, 10, 20, 10, 30, respectively, and that Johnny self-graded them as 15, 0, 10, 20, 0, 30.
   - Johnny needs to change the line in the self-grade file, putting his data for last-name, first-name and email-address, appending (in the same line) comma-separated fields containing all self-grades (non-zeros as well as zeros, as applicable), and ending the line with the field containing the total self-grade for this homework. Therefore, Johnny changes the single line in the self-grading file to the following:

     Doe,Charles,doe_ch@northeastern.edu,15,0,10,20,0,30,75.

5. Prior to submitting your homework, check the self-grading file and make sure that it is completed **exactly** according to **all** of the instructions provided herein. Also, check that you did not overlook any gradable problems.

6. Be sure to submit your completed self-grading file together with your homework.

Please note that a credit of zero may be given for this homework if the self-grading file, completed according to the instructions provided above, is not submitted as part of your homework.

Programming Problems.

Read carefully the following important notes pertaining to all problems in this assignment:

- **In solving all problems below, use the "starter code" skeleton provided with this assignment – see file NNDL_7615_HW4_STARTER_CODE.txt.**
- **MNIST is the dataset that you will use in all problems within this homework.**
- **Note the print() and model.summary() lines in the starter code. You need to submit the results of these lines. This is in addition to the results printed out during the training, and the test-set performance result of the (model.evaluate(ds_tst)) line, all of which you need to submit for each of the problems in this assignment.**
- **Do not forget to use a different name for the "model" for each Programming Problem (in all of the starter code lines). This name should be specific to each Problem (so, if you used the name "model" while working on Problem 1, then you should use another name for the model when you work on Problem 2). In other words, be careful not to override model variables when you go from one Problem to another in this homework.**
- **Problem 1 of this homework is focused on a basic CNN network. As you know, CNN architectures are the appropriate architectures for image tasks (as opposed to MLP architectures). Therefore, Problem 1 represents the right way to address MNIST task and thus constitutes the "main" part of this homework.**
- **The only purpose of Problems 2 and 3 (both focused on MLP architectures), is to serve as an exercise of implementing MLP architectures within the same overall code structure (starter code) using Tensorflow and Keras.**

1 Programming Problem 1:
Simple convolutional neural network (CNN), using Tensorflow and Keras [See point values below].

1.1 Loading dataset and building input pipelines [10 points].

In this part you will:

a) Load MNIST dataset, using tfds "load" function.

b) Build pipelines for training, evaluation, and test data.

Keep in mind the following hints:

1. As you know, MNIST is composed of the training and test sets.
2. Set tfds.load argument shuffle_files to True. It is a good practice to shuffle training data, including larger datasets stored as multiple files.
3. Set tfds.load argument as_supervised to True to return a tuple structure (input, label), i.e., (image, label) in case of MNIST. Otherwise the default (False) will result in a dictionary structure.
4. Set tfds.load argument with_info to True. The info will be part of the 2-tuple returned by tfds.load.
5. Use 10% of the training data as validation set.
6. In input normalization function "normalize_image", remember that you will need the data in as float32, but tfds.load will return them as uint8. To remedy this, use **tf.cast** before dividing by 255.

If your results of the *print* statements for this subsection (as included in the starter code) show tensor shapes (28, 28, 1), you may claim full point-value of this subsection. However, please note that this not guarantee that everything you did in this subsection is correct. Therefore, if your final results of this exercise are not satisfactory, you should re-examine the code you wrote for this subsection as part of your overall debugging process.

1.2 Define CNN network model [10 points].

Use the following very basic CNN architecture, consisting of the following layers in the order listed below:

- Convolution layer 1: 32 kernels (filters), kernel size 3×3, stride=1, zero padding, ReLu activation function.
- Convolution layer 2: 64 kernels (filters), kernel size 3×3, stride=1, zero padding, ReLu activation function.
- Max pooling layer: Use TF default argument values, i.e., MaxPool2D( ).
- Fully connected network consisting of one dense layer FC1 fully-connected to the output layer FC2. All units in the FC1 layer should use ReLu activation functions, and all units in the FC2 layer should use the softmax activation function (you should understand the reason for this). You need to determine the appropriate number of units in FC1 and FC2, based on the nature of the task and on the preceding layers of this particular CNN architecture.

1.3 Compile CNN model [10 points].

Use *SparseCategoricalCrossentropy* loss function, ADAM optimizer with Nesterov momentum, and the *accuracy* metric.

1.4 Train CNN [20 points].

Train for 6 epochs. Remember to use your validation set (not the test set yet). The validation-set accuracy should reach **approximately 0.988**.

1.5 Evaluate test-set performance [30 points].

Evaluate your CNN on the MNIST test set. The test-set accuracy for this architecture should reach **approximately 0.99**.

2 Programming Problem 2:
Basic deep feedforward MLP network, using Tensorflow and Keras [10 points].

Refer to Problem 1 for most of the steps you will need for this problem. Use the same steps as in Problem 1, except for the model definition which follows below.

2.1 Define a simple deep MLP network model.

Use a deep (minimalistic, two hidden layers only) feedforward MLP architecture, consisting of the following layers in the order listed below:

- Hidden layer 1: 300 units, ReLu activation function.
- Hidden layer 2: 100 units, Rely activation function.
- Output layer: The number of units as appropriate for MNIST task, softmax activation function.

2.2 Compile, train, and evaluate this network.

Compile the model using the same settings as those listed in Problem 1 (in Compile CNN model subsection). Train for 6 epochs. Evaluate your two-hidden-layers feedforward MLP network on the MNIST test set. The test-set accuracy for this architecture should be **approximately 0.980**.

3 Programming Problem 3:
Single hidden-layer feedforward MLP network, using Tensorflow and Keras [10 points].

Refer to Problem 1 for most of the steps you will need for this problem. Use the same steps as in Problem 1, except for the model definition which follows below.

3.1 Define a single hidden-layer feedforward MLP network model.

Use a deep (minimalistic, two hidden layers only) feedforward MLP architecture, consisting of the following layers in the order listed below:

- Hidden layer: 100 units, ReLu activation function.
- Output layer: The number of units as appropriate for MNIST task, softmax activation function.

3.2 Compile, train, and evaluate this network.

Compile the model using the same settings as those listed in Problem 1 (in Compile CNN model subsection). Train for 6 epochs. Evaluate your single hidden-layer feedforward MLP network on the MNIST test set. The test-set accuracy for this architecture should be **approximately 0.973**.