CMP-6002A -- Machine Learning

Lab 8 -- Training Neural Networks

In this lab you will continue to explore aspects of the MLPClassifier in the scikit-learn library in Python to train neural networks for recognizing handwritten digits. In the first part of this lab, you will have another look at the misclassified instances and compute the top-k classification accuracy score. Next, you will investigate regularization techniques for training NNs and understand their impacts on the weight matrices of the resulting NNs. In the final part, you are invited/encouraged to code a basic NN class from scratch (i.e. not using scikit-learn), which is strictly optional.

We will continue to use the *MNIST* dataset as detailed in Week 7. Consult Part A of last week's lab sheet if you have not already done so.

Create a new folder "week 8" and copy the Python script file ('lab7_mnistLoader.py') from the "week 7" folder into the new folder and rename it as 'mnistLoader.py'. Make sure the function load_data() can work properly.

Part A. MLPClassifier

In this part you will continue to explore various features of the MLPClassifier (Multi-layer Perceptron classifier) API from the sklearn library. Some of the tasks were listed in Part C of Week 7, so you could reuse/skip them if you have already completed them last week.

To start with, have another read on the official MLPClassifier document. In addition to the parameters and methods you have investigated last week, pay attention to the following ones:

1. *alpha* is the strength of the L2 regularization term.
2. *coefs_* is an import attribute, a list of shape (n_layers - 1,) where the ith element in the list represents the weight matrix corresponding to layer i.
3. *predict_proba(X)* is the method used for probability estimate.

Next, investigate how to save your NN model to a file so that it can be loaded later by another script. Create a new Python script file called 'modelGenerator.py'. Use the *load_data()* function in 'lab8_mnistLoader.py' and the *MNIST* dataset to complete the following tasks:

1. Write a script that allows you to conduct hyperparameter tuning for the models. Test it on the dataset MNIST to report the best performance you can achieve, as well as save the best model, which will be referred to as *bestModel* in this sheet, to a file called 'bestModel.sav'. Can your *bestModel* achieve an accuracy rate of 97% or above for the test subset?

Hint: For saving the NN model in task 1, one may use the *pickle* library; see its official document.

Create a new Python script file called 'modelTopK.py'. In this script:

1. Write a function that can identify all test instances that are misclassified by a NN mode. Test it using the model *bestModel* which will be loaded from 'bestModel.sav' and the test subset.
2. For all misclassified instances from the test subset, print out the probability distribution computed by the *bestModel*.
3. The top-k classification accuracy is one of the core metrics in machine learning. Here, k is conventionally a positive integer, such as 1 or 5. For instance, to evaluate top-5 accuracy, the classifier must provide relative likelihoods (e.g. probability distribution) for each class. When these are sorted, a classification is considered correct if the correct classification falls anywhere within the top 5 predictions made by the network. Write a function to compute top-k classification accuracy for a NN model on a test set and a given parameter k.
4. Test your function in Task 4 on your *bestModel* and the test subsets, with k ranging from 1 to 5.
5. [Optional] Compare the top-k scores with the normal accuracy score by plotting them on a figure (as you design).

Hint: Optional tasks are not required for completing the other parts, so you may return to it after completing tasks in the other parts.

Part B. Regularization

In this part you will investigate regularization techniques for training NNs and understand their impacts on the weight matrices of the resulting NNs. Create a new Python script file called 'nnWeights.py'.

1. The 1-norm of a matrix is the maximum absolute column sum of the matrix; see, e.g. wikipedia
2. Write a function whose input is a NN model (as an instance of the MLPClassifier) and whose output is the list of the 1-norms of the weight matrix of each layer. Test it on your NN *bestModel*.
3. Create a MLPClassifier NN model with the same architecture as the one you used in

bestModel (but not the same weights). Now train it with the regularization technique L2 weight decay with decay coefficients 0.1. Compare the accuracy, as well as the 1-norms of the weight matrix of each layer, between this model and that of *bestModel*.
4. Repeat task 3 with different decay coefficients, say 0.01 and 0.001.
5. [optional] Collect your results from 3 and 4 to a table and save it to a file.
6. [optional] Explore other regularization techniques mentioned in the lecture slide.

Hint: You may use the matrix norm function from NumPy.

The letter λ is used for the weight decay coefficient in the lecture slides, but in MLPClassifier the attribute is called *alpha*.

Part C. Optional Topics

This part is optional. To further improve your intuition and understanding of MLP, you may further complete one or more optional tasks in turn as below.

1. Design and implement a Python class called *NNetwork* to represent a neural network to conduct the task of digital recognition without using the scikit library. Your model should at least contain a method which read a picture from the MNIST dataset and output a digit. You don't need to worry about how to train it at this stage.
2. Load the MLPClassifier model you saved in the file 'bestModel.sav' and use the relevant parameters to initialize an instance of *NNetwork*.
3. Test your own neural network on the MNIST dataset, and compare it with that of *bestModel*.
4. Investigate which additional components are required for your *NNetwork* class so that it can be trained.

Hint: You may consult the code listed in Chapter 2 of the book **Neuro Networks and Deep Learning** listed in the reference section.

Reference Materials

> **YouTube videos**, Chapter 1 to 4 in a 3Blue1Brown series. The first one can be found at https://www.youtube.com/watch?v=aircAruvnKk&t=908s
>
> **Neuro Networks and Deep Learning**, An online book by Michal Nielson http://neuralnetworksanddeeplearning.com/
>
> **NMIST Dataset**, https://en.wikipedia.org/wiki/MNIST_database
>
> **Neural Networks model at scikit-learn**, https://scikit-learn.org/stable/modules/neural*networks*supervised.html