Advanced Machine Learning – Assignment 3

Group Assignment

In this assignment, we will explore neural network models using the Fashion-MNIST data (https://www.tensorflow.org/datasets/catalog/fashion_mnist). First, check out its document and download the data. Since it has no validation data, we need to split a proportion of its training data for validation purpose, say, 25% of training samples as our validation set.

Also, you may find this blog helpful in finishing this assignment (https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/). The blog also shows how to load the data using the keras package.

First, you need to preprocess the data by normalising the pixel values of grayscale images, e.g. rescale the pixel values from [0,255] to the range [0,1]. See the "Prepare Pixel Data" section in the above blog post.

Then, you will implement the kernelised SVM for benchmark purpose. Before doing that, we need one further preprocessing of the dataset. Specifically, we want to get a sense of how hard this classification task is. Note that the image data is two-dimensional as the example plots in the blog post, while machine learning models such as kernelised SVM can only handle one-dimensional input. So you need to flatten the matrix into vector using the flatten function in numpy (https://numpy.org/doc/stable/reference/generated/numpy.matrix.flatten.html). There are multiple ways to flatten the data; for this assignment, we will follow the default option of the numpy function and adopt the row-major (C-style) order.

(a) Implement kernelised SVM for the dataset and report your model's accuracy on the test set. You need to tune the hyper-parameter (with the validation data) for the kernelised SVM model, including the parameter C and the choice of kernel. Explain your procedure of buiding the model.

(b) Implement a fully-connected neural network with the same setup as the code in Activity 6.5. You might need to adapt the model architecture therein to fit in this task, such as the input and output dimensions. For intermediate layers, you can simply use 12 and 8 neurons as in the exercise code. Report the test accuracy of your trained neural network model. For this part, since we don't tune the hyper-parameter, you can combine the training and validation set for the training of the neural network.

(c) Now, let's explore a bit in terms of the hyper-parameter tuning of the neural network models. Please try out the following options and briefly report what you observe:

- Add one more layer between the 12-neuron layer and the input layer; decide the numbers of neurons on that layer on your own.
- Change the numbers of the neurons (12 and 8) to other values.
- Change the loss function (through the *loss* in the *compile* function).
- Change the *epochs* and *batch_ size* in the fit function.

Will you be able to improve the performance of the neural network model? You should follow the training-validation-test procedure – to decide your selected values of the hyper-parameters through the validation set and then report the test error on the test set.

Finally, run the *F_MNIST.py* (extracted from the blog as a basic version of the convolutional neural network (CNN).) The advantage of CNN is that it can directly handle 2-dimensional (or even higher-dimensional) input so that there is no need to flatten the data. As we can imagine, the flattening of the image data will destroy the structural information only contained in the 2-dimensional data. That is the primary reason why CNN usually performs better than the fully-connected neural networks on image tasks. We didn't cover in details the model of CNN and hope to use this example as a starting point for your study of more advanced neural network models. For here, you can treat the model of CNN as a "black box" and try to understand the model from reading the code and running experiments.

(d) Run the *F_MNIST.py* and briefly explain what the following functions do in the script

- define_model
- evaluate_model
- summarize_diagnostics

Briefly explain how the validation (for the training of CNN) is set up in this script. What is the output of the script, and how does it compare to the results in the previous parts?