Machine Learning CW3

Objective. The goal of this coursework is to gain a hand on experience with multilayer Perceptrons (MLPs) by implementing them from scratch and applying them to toy classification tasks.

In the final CW, we will use the outcome of this CW as well as other machine learning algorithms to solve more realistic problems.

Problem setting. We will use a binary classification problem with two-dimensional inputs: You will be given a training set $X_{\text{tr}} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\} \subset \mathbb{R}^2 \times \{0, 1\}$, a test set $X_{\text{te}} = \{(\mathbf{x}'_1, \mathbf{y}'_1), \ldots, (\mathbf{x}'_M, \mathbf{y}'_M)\} \subset \mathbb{R}^2 \times \{0, 1\}$, and a separate set of inputs $X_{\text{grid}} = \{\mathbf{x}''_1, \ldots, \mathbf{x}''_L\} \subset \mathbb{R}^2$ presented as a regular grid sampling of $[-15, 15] \times [-15, 15]$. Each input in the training and test sets is accompanied by the corresponding class label (either 0 or 1).

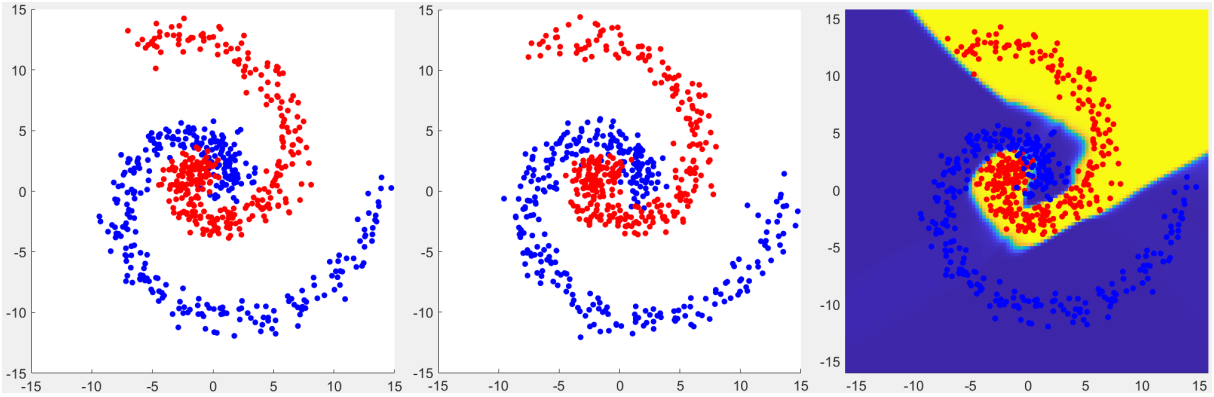

Figure 1: Our toy classification problem: (left) training set X_{tr} (class 0 and 1 are respectively highlighted in red and blue); (middle) test set X_{te}; (right) outputs of the trained MLP on X_{grid} overlaid with the training set X_{tr}. The average training and testing accuracies of the final model are 98.25% and 97.78%, respectively.

Task. Our task is to train an MLP on X_{tr}, apply the trained MLP to X_{te} and X_{grid}, and visualize the outputs of the MLP model X_{grid}. Our data are shown in Fig. 1: The training and test sets respectively consist of 630 pairs of inputs and the corresponding ground-truth outputs: In Fig. 1(left) and (middle), the location of each dot represents the corresponding training input (x^1, x^2-coordinate values) while the class labels are highlighted by the color. The accompanying 'Trn.csv' file contains this training set: For each line, the first two columns represent an input $\mathbf{x} = [x^1, x^2]^\top$ and the third column provides the corresponding class label (0 or 1). Similarly, 'Tst.csv' file provides the test set. The grid set X_{grid} consists of 10,200 data instances ('Grid.csv' file) where each line (consisting of two columns) represents an input.

What to hand on? Please submit your code and a report (up to two pages):

Your code should implement the training and test phases of an MLP. Please use the fully-stochastic gradient descent for training. The submitted code package should include a python script 'mymlp.py' which

1. reads X_{tr} ('Trn.csv') from the directly where the script is executed;
2. trains an MLP on X_{tr};
3. reads X_{te} ('Trn.csv') and X_{grid} ('Grid.csv') from the same directory;

4. applies the trained MLP to X_{te} ('Tst.csv') and X_{grid} ('Grid.csv'), and measure the average prediction accuracy (in %) on X_{te}.

Your code should meet the **requirements stated in the accompanying 'README.md' file**: The code submission should specify the python version and required packages. If your code package contains more than one file, please make sure that all four steps mentioned are performed by executing 'mymlp.py': 'mymlp.py' can use other scripts. If your code does not meet the code requirements, the final mark will be reduced to 50%.

You can use basic math libraries e.g. NumPy. However, you should implement the training steps of MLP from scratch. In particular, the calculation of the error gradient for backpropagation has to be implemented by yourself. If you use any existing backpropagation code or automatic differentiation e.g. provided by TensorFlow, you will get 0 mark for your code.

Your report should

1. specify the hyperparameters used in your experiments (e.g. the number of layers, number of neurons in each layer, and learning rate scheduling scheme);
2. report the accuracies of the trained model on X_{tr} and X_{te};
3. visualize the outputs of the model on the grid set X_{grid}. This visualization will look similar to the third column of Fig. 1 while the shapes of the decision boundaries formed by the trained MLPs might vary depending on the choice of the hyperparameters.

In addition, the report should

4. present the results obtained by training a separate MLP model on the first 40 data entries of X_{tr} (corresponding to the first 40 lines of 'Trn.csv'): Provide the accuracies of the resulting new model on X_{tr} and X_{te};
5. visualize the outputs of 4) on X_{grid}. Check if in this case, your new model achieves 100% training accuracy (on the first 40 data points of X_{tr}): This shows how models can overfit when training data is limited.

Please type your report: If your submission is not typed (e.g. a scanned pdf of handwritten solutions), the final mark will be reduced to 50%. Please submit a single zip file that contains your code and a pdf document of your report. Please format the submission as in **'StudentID_Name.zip'**, e.g. 20221234_KwangInKim.zip. If your submission does not meet this file name requirement, the final mark will become 80% of the initial mark.

Grading.

1. Code (70/100): We will simply check if your code runs as specified. If your code successfully implements the training and testing phases of an MLP and it meets all requirements listed in the previous paragraphs, you will earn full marks.
2. Report (30/100): We will check if your report appropriately visualizes the outputs obtained by the two trained MLP models and if all hyperparameter values are specified.

Verifying the gradients. Calculating the gradient of an MLP training objective (e.g. sum of losses) is mathematically straightforward. However, verifying the correctness of the corresponding software implementation (i.e. backpropagation) can be burdensome. In this case, comparisons with numerical gradients can help.

If defined, the gradient $\nabla f(\mathbf{x})$ of a function $f : \mathbb{R}^n \to \mathbb{R}$ at point $\mathbf{x}$ is given as a vector consisting of partial derivatives:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x^1}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x^n}(\mathbf{x}) \end{pmatrix}.$$

The i-th partial derivative $\frac{\partial f}{\partial x^i}(\mathbf{x})$ at $\mathbf{x}$ is defined as

$$\frac{\partial f}{\partial x^i}(\mathbf{x}) = \lim_{\epsilon \to 0} \frac{f(\widetilde{\mathbf{x}}^i) - f(\mathbf{x})}{\epsilon} \tag{1}$$

where $\widetilde{\mathbf{x}}^i = [x^1, \ldots, x^i + \epsilon, \ldots, x^n]^\top$. Often, $\frac{\partial f}{\partial x^i}(\mathbf{x})$ can be well-approximated by fixing ϵ at a small value instead of actually calculating the limit in Eq. 1:

$$\frac{\partial f}{\partial x^i}(\mathbf{x}) \approx \widetilde{\frac{\partial f}{\partial x^i}}(\mathbf{x}) = \frac{f(\widetilde{\mathbf{x}}^i) - f(\mathbf{x})}{\epsilon}.$$

The numerical gradient $\widetilde{\nabla} f(\mathbf{x})$ of f at $\mathbf{x}$ is then defined as

$$\widetilde{\nabla} f(\mathbf{x}) = \begin{pmatrix} \widetilde{\frac{\partial f}{\partial x^1}}(\mathbf{x}) \\ \vdots \\ \widetilde{\frac{\partial f}{\partial x^n}}(\mathbf{x}) \end{pmatrix}.$$

Figure 2 shows an example of analytical gradient (constructed via backpropagation) and the corresponding numerical gradient. The respective graphs are well-aligned assuring that backpropagation is correctly implemented.

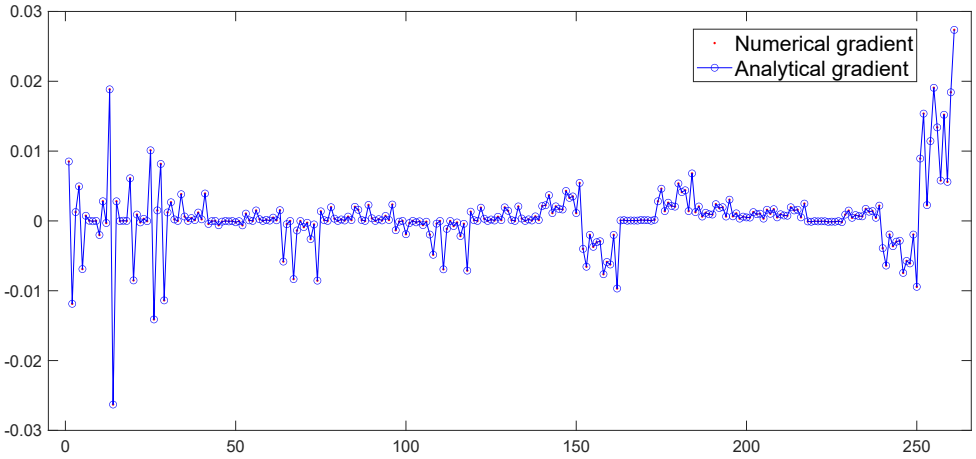

Figure 2: An example evaluation of analytical gradient of an MLP objective function with respect to its weight vector (combining all MLP weights) and the corresponding numerical gradient with $\epsilon = 10^{-5}$. The network consists of an input layer of size 2, three hidden layers of size 10 each, and an output layer of size 1. Each neuron employs an offset and therefore, the total number of weights is 261 ((2+1)×10+(10+1)×10+(10+1)×10+(10+1)). We used the fully-stochastic gradient descent.