Deep Learning (FSS22)
Assignment 1: Neural Networks

The archive provided to you contains this assignment description, datasets, as well as Python code fragments for you to complete. Comments and documentation in the code provide further information.

It suffices to fill out the "holes" that are marked in the code fragments provided to you, but feel free to modify the code to your liking. You need to stick with Python though.

Please adhere to the following guidelines in all the assignments. If you do not follow those guidelines, we may grade your solution as a FAIL.

Provide a single ZIP archive with name `dl22-<assignment number>-<your ILIAS login>.zip`. The archive needs to contain:

- A single PDF report that contains answers to the tasks specified in the assignment, including helpful figures and a high-level description of your approach. Do not simply convert your Jupyter notebook to a PDF! Write a separate document, stay focused and brief. As a guideline, try to stay below 10 pages.
- All the code that you created and used in its original format.
- A PDF document that renders your Jupyter notebook with all figures. (If you don't use Jupyter, then you obviously do not need to provide this.)

A high quality report is required to achieve an EXCELLENT grade. Such a report is self-explanatory (i.e. do not refer to your code except for implementation-only tasks), follows standard scientific practice (e.g. when using images, tables or citations), does not include on hand-written notes, and does not exceed 10 pages. In addition, label all figures (and refer to figure labels in your write-up), include references if you used additional sources or material, and use the tasks numbers of the assignments as your section and subsection numbers.

Hand-in your solution via ILIAS until the date specified there. This is a hard deadline.

Rule: Use PyTorch

In this assignment, we explore simple feedforward networks in PyTorch. To get used to PyTorch, you **must not use NumPy** in your solutions to this assignment. Instead, **use PyTorch** (and plain Python, of course) throughout.

1 Perceptron Learning

We look at binary classification using a perceptron. We use two 2D datasets: a separable dataset $\mathcal{D}_1$ (`X1` and `y1`) and a non-separable dataset $\mathcal{D}_2$ (`X2` and `y2`). Both datasets include an additional bias feature.

a) Complete function `pt_train` to train a perceptron using Rosenblatt's perceptron learning algorithm (without an explicit bias term). Write the function yourself, i.e., do not use any existing implementations. A description of the parameters and expected result can be found in the Python file provided to you.

   - Your function should print the total number of examples tested (both correctly or incorrectly classified) as well as the total number of weight updates.
   - Write your function such that it tests the training examples in the order in which they appear in the training set. As usual, we refer to one complete pass through the training set as an *epoch*. After an epoch completes, the training set is processed from the beginning again in the subsequent epoch.
   - Function `pt_train` takes argument `maxepochs` for the maximum number of epochs to run. When is it safe to stop perceptron learning earlier? Why? Implement this early stopping in your function.

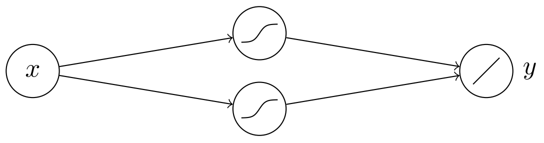
b) Test your function on $\mathcal{D}_1$ and $\mathcal{D}_2$. You can plot the decision boundaries and print the misclassification rates of multiple runs of your training algorithm as well as a linear SVM and a logistic regression classifier (code provided). Discuss and explain the results.

c) Extend your function such that it can also train a perceptron using the pocket algorithm (argument `pocket=True`). In each epoch, your function should process N random examples sampled with replacement (try `torch.randint(N,(N,))`).

d) Test your implementation of the pocket algorithm on $\mathcal{D}_1$ and $\mathcal{D}_2$. Are the results different than before? Discuss and explain.

2 Multi-Layer Feed-Forward Neural Networks

In this task, we look at regression using multi-layer feedforward networks (FNN) of form



where we vary the number of units in the hidden layer. We use a 1-dimensional dataset $\mathcal{D}_3$ (`X3` and `y3`, as well as `X3test` and `y3test`).

a) Look at the training data and conjecture how a fit for an FNN with zero, one, two, and three hidden neurons would look like.

b) The Python file provides code for training based on Scipy's BFGS optimizer. Train an FNN with two hidden neurons, determine the mean squared error (MSE) on the training and the test data, and plot. Is the result as you expected? Now repeat training multiple times. What happens? Explain.

c) Train a FNN with 1, 2, 3, 10, 50, and 100 hidden neurons. In each case, determine the MSE on the training and the test dataset. Then plot the dataset as well as the predictions of each FNN on the test set into a single plot. What happens when the number of hidden neurons increases? Is this what you expected? Discuss!

d) Train a FNN with 2 hidden neurons and visualize the output of the hidden neurons (=distributed representation, code provided). Then visualize the output of the hidden neurons scaled by the weight of their respective connection to the output (code also provided). (As before, you may want to repeat this process multiple times.) Now repeat with 3 hidden neurons, then 10 hidden neurons. Try to explain how the FNN obtains its flexibility. Is the distributed representation intuitive?

e) (Optional.) Repeat the above experiments with different optimizers; e.g., try the Adam optimizer provided with PyTorch (code provided). Also experiment with different numbers of hidden layers and/or with different types of units in the hidden layers. For example, it is instructive to repeat subtasks a)—d) with ReLU units. Discuss.