Deep Learning (FSS24)
Assignment 1: Neural Networks

University of Mannheim

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. Please note the following:

- 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 `dl24-<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. You must **stay below 10 pages**, excluding references and appendix.
  - 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 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.
- You **may work on this assignment in pairs**—i.e, with one (and only one) additional student—and then hand in a pair submission. To do so:
  - The PDF report and notebook must clearly report then **name and ILIAS login** of **both students** right at the beginning.
  - **Both students must submit** the same assignment on ILIAS separately.

  To be clear, if only one student of the pair submits the assignment, then only this student will receive the grade. 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 Implement an MLP

In this first task, you will implement an MLP (with the standard template architecture) manually in PyTorch.

a) Examine the class `LogisticRegression` provided to you. Pay attention to (i) how the parameters are stored, (ii) how the calculations in the forward pass are performed, and (iii) how the model is initialized and used.

b) Complete the implementation of the class `MLP` provided to you. The expected functionality is documented directly in the Python file.

   Requirements. You must implement the MLP without using pre-canned PyTorch neural network functionality. In particular, you must not use any layer from `torch.nn`.[1] You can (and should) use `torch.nn.Parameter` and `torch.nn.Module`, however.

c) Expand the implementation of the class `MLP` such that

   (i) When a vector ($\boldsymbol{x}$) is provided, it computes the network's output ($\boldsymbol{y} = f(\boldsymbol{x})$) as a vector.

   (ii) When a matrix is provided, it computes the network's output for each row as a matrix. I.e.,

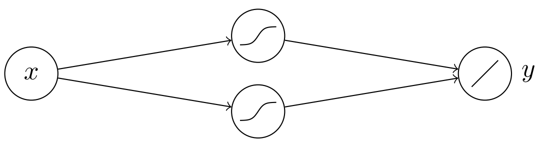
$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_N^\top \end{pmatrix} \to \begin{pmatrix} f(\boldsymbol{x}_1)^\top \\ \vdots \\ f(\boldsymbol{x}_N)^\top \end{pmatrix} = \boldsymbol{Y}.$$

   Models are commonly implemented like this in PyTorch: the first dimension (*batch dimension*) corresponds to an example, the remaining dimensions then hold the actual values of that example.

   Requirements. Do not use `torch.vmap` in your solution, but directly implement the required operations efficiently.

2 Experiment with MLPs

In this task, we look at regression using MLPs of form



where we vary the number of units in the hidden layer. Luckily, you implemented such networks in the previous tasks already.

We use a 1-dimensional dataset $\mathcal{D}_1$ (`X1` and `y1`, as well as `X1test` and `y1test`).

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!

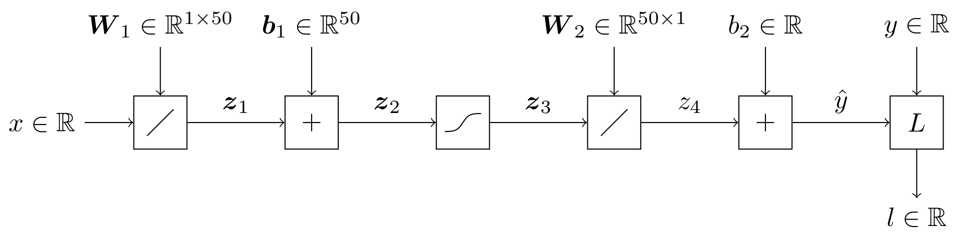
[1] PyTorch provides pre-canned layer implementations in classes such as `torch.nn.Linear` (see the documentation for an overview). We'll cover those later in the course; here we stick to the basics.

d) Train a FNN with 2 hidden neurons and visualize the output of the hidden neurons (=distributed representation / embedding, 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.

3 Backpropagation

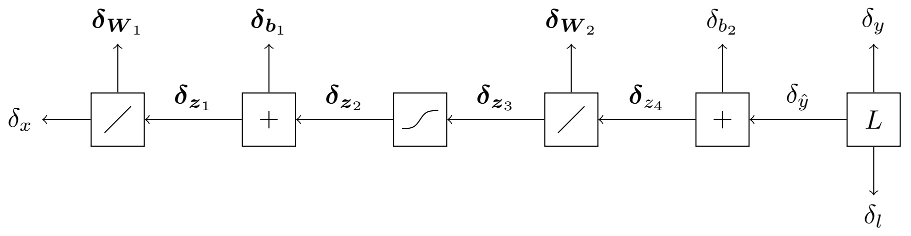
Consider the network of the previous task with a single hidden layer of size $Z = 50$. In this task, we are going to compute various gradients of the network output with respect to its inputs and weights manually. The network is represented by the following compute graph:



The $\boxed{\diagup}$ operations perform matrix-vector multiplications (e.g., $\boldsymbol{z}_1 = \boldsymbol{W}_1^\top x$), the $\boxed{+}$ operations element-wise addition (e.g., $\boldsymbol{z}_2 = \boldsymbol{z}_1 + \boldsymbol{b}_1$), and the $\boxed{\sim}$ operations applies the logistic function element-wise (e.g., $[\boldsymbol{z}_3]_k = \sigma([\boldsymbol{z}_2]_k)$). Unit $\boxed{L}$ computes the loss between prediction y and target $\hat{y}$:

$$L(y, \hat{y}) = (y - \hat{y})^2.$$

The backward graph for this network looks as follows:


$\boldsymbol{\delta}_{\boldsymbol{W}_1}$
$\boldsymbol{\delta}_{\boldsymbol{b}_1}$
$\boldsymbol{\delta}_{\boldsymbol{W}_2}$
δ_{b_2}
δ_y
δ_x
$\boldsymbol{\delta}_{\boldsymbol{z}_1}$
$\boldsymbol{\delta}_{\boldsymbol{z}_2}$
$\boldsymbol{\delta}_{\boldsymbol{z}_3}$
$\boldsymbol{\delta}_{z_4}$
$\delta_{\hat{y}}$
L
δ_l

We provide code to extract the model parameters from a trained model, run the model using PyTorch on a single input `x`, and extract (some of) the gradients using PyTorch's autograd.

a) Compute the results of the forward pass directly using basic PyTorch operations. Your code must only access model inputs, targets, and parameters (i.e., `x`, `y`, `W1`, `b1`, `W2`, `b2`).

b) Compute the results of the backward pass directly using basic PyTorch operations (i.e. without using autograd). Your code must only access model inputs, targets, parameters, and forward pass outputs (i.e., `x`, `y`, `W1`, `b1`, `W2`, `b2`, `z1`, `z2`, `z3`, `z4`, `yhat`). In your report, give derivations and resulting formulas (not Python code) for each of these quantities.