10907 Pattern Recognition

Lecturers
Prof. Dr. Thomas Vetter ⟨thomas.vetter@unibas.ch⟩
Prof. Dr. Ivan Dokmanić ⟨ivan.dokmanic@unibas.ch⟩
Dr. Dennis Madsen ⟨dennis.madsen@unibas.ch⟩
Dana Rahbani ⟨dana.rahbani@unibas.ch⟩

Tutors
Martin Ramm ⟨martin.ramm@unibas.ch⟩
Viktor Gsteiger ⟨v.gsteiger@unibas.ch⟩

Exercise 5 — Neural Networks

Deadline **05.12** Upload .ZIP on Adam.

The exercises are done in groups of 2 students. Only upload 1 version of the exercise on Adam and specify your group partner.

Upload in .zip format containing 1 .pdf file with your answers to the questions and the code folder files (do NOT include the data folder). Create the .zip file with the `createSubmission.py` script.

-0.5 Point for uploading in wrong format!

In this exercise, you will compute one iteration of the gradient descent algorithm by hand. In addition, you will implement the backpropagation and gradient descent methods in PyTorch. Finally, you will build simple neural networks for 2D toy dataset and binary image classification tasks.

Data:

The provided skeleton code mostly loads the data for you and reshapes it into an appropriate format. For the *Binary Image classification* task, the data loader also need to be specified.

- `Parabola` ($d = 2$): A very small 2D dataset. Each `npy`-file contains the training and testing sets for a problem. Use it for developing the algorithms and studying the behaviour of your neural networks. The files are named `(train|test)_(inputs|targets)`.
- `Flower` ($d = 2$): All training and test data are found in the file `flower.mat`.
- `horse_not_horse` ($d = 32 \times 32 \times 3 = 3072$): An extract from the popular CIFAR dataset. The raw image files are provided in the subfolder `img/horse_no_horse` with subfolder for `training` and `validation`. 5000 training images for each class is provided for training and 500 for validation.

1 Gradient Descent and Backpropagation (1 iteration - by hand)

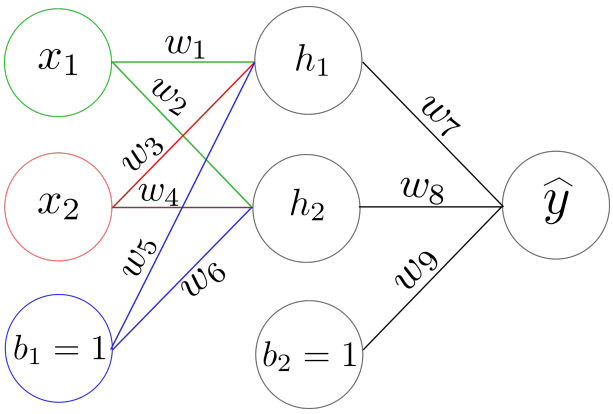

Figure 1: Fully connected Neural Network with three layers.

Update the parameters of the neural network in Figure 1 with the gradient descent algorithm based on one data point $X = [1, 1]$ with label $y = 1$. Consider the following network parameters for your calculations:

- Weights: $w_1 = 0.7$, $w_2 = 1.3$, $w_3 = 1.5$, $w_4 = 0.1$, $w_7 = 0.7$, $w_8 = 0.8$ and all bias weights set to 0
- The hidden neurons use a sigmoid activation function:

$$a(t) = \frac{1}{1 + e^{-t}}$$

- The output neuron is computed with a simple linear activation function.

(a) **Forward Pass**
Calculate the values of the hidden neurons h_1, h_2 and the output neuron $\hat{y}$ and evaluate the error $\mathcal{L}$ of the prediction result. Use the mean squared error: $\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$.

(b) **Backward Pass and Parameter Update**
Calculate the partial derivative (with backpropagation) of the prediction error $\mathcal{L}$ w.r.t. each weight $\frac{\partial \mathcal{L}}{\partial w_i}$ and update the weights via: $w_i^* = w_i - \eta \frac{\partial \mathcal{L}}{\partial w_i}$ with $\eta = 0.2$. Use the chain rule for obtaining the partial derivatives. Be aware that the derivative of the sigmoid function is: $a(t)' = a(t) * (1 - a(t))$.
`HINT: Verify your result by testing if the prediction error decreases after the weight update.`

2 Automatic Differentiation with PyTorch

In this exercise, you have to implement the neural network depicted in Figure 1 in PyTorch and verify your backpropagation calculations from the previous section. The helper script for this task is provided in `ex5_NN_1_Toy.py`. Implement the forward computation as scalar products according to the *Manual Linear Regression* example presented in the lecture notebooks on neural networks. Invoke the automatic differentiation by calling the ".backward()"-function on your error variable.
Verify that the computed gradients are the same as the ones you obtained in your backpropagation implemention results.
`Note: Do NOT use the PyTorch implemented optimization algorithms and loss functions (torch.optim and torch.nn).`

`In the following exercises you can use the full functionality of PyTorch.`

3 Classification with a Multi-Layer Perceptron

In this exercise, you have to design a Deep-Neural-Network for 2 simple 2D classification problems. The template code can be found in `ex5_NN_2_Parabola.py` and the network to be implemented in `mySimpleNN.py`. Design a network which is expressive enough to be used on both the *Parabola* data and the *flower* data. You can use the `Trainer` class in the file `network2DHelper.py` which already implements all functionalities needed to train and test a neural network as well as for visualizing the decision function. Your tasks are as follows:

- Define the neural network in the `mySimpleNN` class. It is up to you how complex the network should be, as well as how many hyperparameters you are introducing (regularization such as dropout, initialization, activation function, normalization, batch-normalization etc.).
- Draw your network as a computational graph.
- Train the network such that 'good' decision boundaries are obtained for both the training and testing datasets. The `Trainer` class will automatically output plots with the decision boundaries under *output/*.

```
Note: Use the BCELoss (criterion) to be compatible with the Trainer class!
CrossEntropyLoss in PyTorch is implemented for multiple class output (1 output
per class). BCELoss instead works with 1 output like logistic regression,
where the output is the probability of belonging to one of the classes.
```

4 Binary Image Classification

In this exercise you have to implement 3 different neural networks to classify images of *horses* or *not-horses*. The data is from the *CIFAR-10* dataset which consist of 10 different classes of images (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) of dimension 32x32x3. You will be working on a subset of the dataset where we have the full *horse* dataset and the *not-horse* is a mixture of the other 9 classes. The data loading can be configured in the `myNetworkTrainer.py` file to add data augmentation.
Design 3 neural networks in the `myImageNN.py` file:

- MyLogRegNN: Design a logistic regression classifier as a neural network. *Hint: To avoid converting your images into 1D vectors for the LogReg and DNN networks, instead use the x.view(-1, dim) function in the forward pass.*
- MyFullyConnectedNN: Similar to the *mySimpleNN* task above - design a fully connected neural network.
- MyCNN: Design a neural network with convolutional layers.
  Note: Even though the image has 3 dimension, a Conv2d module is the function to use on images, where we specify the in_channels=3.

Also here you are allowed to use all kind of hyperparameters to improve your performance.

```
Hint: The summary function from torchsummary can be used to check the size of
your defined networks. Specify the image dimension as a tuple:
summary(model, (3, 32, 32))

Use a sigmoid function as the last unit in each of your networks to be
comparable: y_hat = torch.sigmoid(_)
```

5 Theory / coding questions [10p]

1. What part of a neural network makes it able to estimate non-linear functions? List two examples. [0.5P]
2. Explain the "vanishing gradients" problem and list one approach to minimize the risk of its occurrence. [0.5P]
3. For the backpropagation by hand, please specify (and include your computations): [4P]
   - Network loss $\mathcal{L}$
   - Partial derivative of $\mathcal{L}$ w.r.t. each weight, i.e. $\frac{\partial \mathcal{L}}{\partial w_i}$
   - Weight update when using $\eta = 0.2$
   - Network loss $\mathcal{L}$ with the updated network weights
4. Implement the simple network in `ex5_NN_1_Toy.py` and train the network for 10 iterations with $\eta = 0.2$. Specify the loss for each iteration. [0.5P]
5. Implement the simple network in `ex5_NN_1_Toy.py` and train the network for 10 iterations with $\eta = 1.0$. Specify the loss for each iteration. [0.5P]
6. Draw your network as specified in `mySimpleNN.py` as a computational graph. [1P]
7. Show images of the decision surfaces for the `flower` and `parabola` datasets (both training and test) after the final iteration (images are dumped under `output/flower` and `output/parabola`). [1P]
8. Train your LogReg (`ex5_NN_3_Images_LogReg.py`), FCN (`ex5_NN_3_Images_FCNN.py`) and CNN (`ex5_NN_3_Images_CNN.py`) networks. Show the `_accuracy_curve` plots for each network training as output under the sub-folder `output/` and specify the number of trainable parameters for each of the 3 networks[1.5P].
9. Which network architecture would you in general choose for image classification? And why? [0.5P]