10907 Pattern Recognition

Lecturers
Prof. Dr. Thomas Vetter ⟨thomas.vetter@unibas.ch⟩

Assistants
Dennis Madsen ⟨dennis.madsen@unibas.ch⟩
Dana Rahbani ⟨dana.rahbani@unibas.ch⟩
Moira Zuber ⟨moira.zuber@unibas.ch⟩
Genia Böing ⟨genia.boeing@unibas.ch⟩

Exercise 5 — Neural Networks

Introduction	18.11
Deadline	**26.11** Upload code to Courses.
	25.11+26.11 Group presentations, U1.001, see schedule online

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.

You can download the data needed for this exercise and a code skeleton from the following repository: `https://bitbucket.org/gravis-unibas/pattern-recognition-2019`.
A number of helper functions are provided - your task is to implement the *TODO* parts in the python (.py) files.

Remember to upload your code to courses in a ZIP file. Do NOT include the data folder. Only the python files you edited should be included + a file containing the group members names. Only 1 person from the team should upload!

Beside code upload, remember to upload the classification report with performance table, plots and network architecture drawings.

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.

Gradient Descent and Backpropagation (1 iteration - by hand)

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: $w_1 = 0.5$, $w_2 = 0.3$, $w_3 = 0.3$, $w_4 = 0.1$, $w_5 = 0.8$, $w_6 = 0.3$, $w_7 = 0.5$, $w_8 = 0.9$, $w_9 = 0.2$.
- The hidden neurons use a sigmoid activation function:

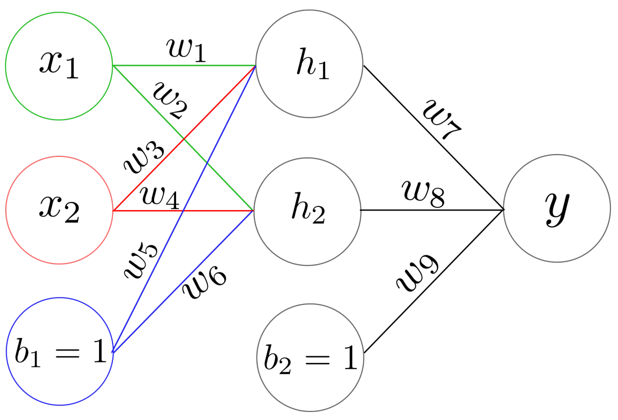

Figure 1: Fully connected Neural Network with three layers.

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

- 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 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: $a(t)' = a(t) * (1 - a(t))$.
Verify your result by testing if the prediction error decreased after the weight update.

1 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.`

2 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 `trainer.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 *Training* class will automatically output plots with the decision boundaries under *output/*.

```
Note:  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.
```

3 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 and model training is to be imlemented in the `ex5_NN_3_Images.py` file.
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. As it would be infeasible to train the networks during the quizzes, you will instead have to hand-in a small performance overview together with your code.
The report needs to contain:

- Overview table of final error and accuracy for all 3 classifiers:

	Training Error	Validation Error	Training Acc.	Validation Acc.	#Epochs
LogReg					
DNN					
CNN					

- Accuracy and error plots for all 3 networks (epochs on the x-axis). The plotting functionality and figure saving can be found in the *writeHistoryPlots* function.
- Neural network architecture drawings for each of the models. Each layer in the drawing should contain details such as kernel-size, stride, padding, number of neurons, activation function.
- Count of the total number of trainable parameters in each of your networks.

- Regularization section describing the regularization techniques you have made use of, e.g. dropout, augmentation, batch-normalization.
- Learning algorithm and hyper-parameter values (e.g. learning-rate).

`Hint: Make use of the model training code from the Transfer-learning notebook and the introduction notebook about custom nn modules. Also note the summary function from torchsummary which can be used to check the size of your trained networks.`