Exercise in Big Data Anaylytics
Prof. Dr. Youness Dehbi
Computational Methods Lab
HafenCity University Hamburg

# Exercise 2: First neural networks

Hand-out : Wednesday, 19.04.2023
Hand-in : Wednesday, 03.05.2023

## General Notice

The exercises can be solved in groups of *two to three* students.Answers to theoretical questions and the source code of the programming exercises and result files should be presented on demand.

## Regression Network

The first neural network you develop will be a regression network. For this task, you receive an incomplete source code **main_regression_who_colab.ipynb**, with which life expectancy can be estimated on the basis of certain features collected from the WHO. In the source code, gaps, which are marked with **TODO****, have to be filled and theoretical questions have to be answered. For experimentation, further changes to the code should be made at the end, if requested. The following questions guide you step by step through the code:

At the beginning the required modules are imported. In addition, it is checked whether a graphics card (GPU) is available. Then the data set is read in.

1. **(2)** Load the data set "Life Expectancy Data.csv" and print its size, which is the number of samples and its dimension, which is the number of features per sample. It may be necessary to adjust the path for reading the data.

2. **(2)** Divide the data set into input x and target y features. For a start, the BMI column shall be used as x and the column with the life expectancy as y.

In the following x and y are normalized, which is very important for the stability of neural networks. If several input features were used, those in lower value ranges would possibly be suppressed without normalization.
They are also converted to torch tensors x $\rightarrow$ X and y $\rightarrow$ Y, so that they can later pass through the network.

3. **(3)** Divide X and Y into a training, validation and test set. Determine the division of the sets appropriately. The proportions must add up to 1. Explain, why a split in training, validation and test set is important.

Now we can start designing the architecture of our regression network.

4. **(4)** Look at the RegressNet class. It consists of 2 methods. __init__(self, inputSize, outputSize) specifies the layers that this class holds. forward(self, x) specifies the connection between these layers. Note that activation functions are also used here. Describe the architecture of the network with the technical vocabulary you learned in the lecture.

5. **(3)** Consider which input and output dimensions the network needs to have in the current case. Afterwards create an instance net of RegressNet.
   Hint: Think about the dimensions of X and Y.

If a graphics card is available, X, Y and net are now sent to the GPU for efficient computation. Otherwise the network is trained on the CPU.

6. **(2)** In order to train the network, the so-called hyperparameters must be defined. In this case it is the number of epochs (num_epoch) the network should be trained, the learning rate (learn_rate), the loss function and the optimizer. The latter two have already been defined. We use a mean squared error loss function MSELoss() and ADAM optimizer.
   Hint: The learning rate is usually in a range from 1e-1 to 1e-5.

Now follows the training. In a loop over the epochs, the training data passes through the network, which is called forward pass. The training loss is calculated from the forward pass prediction and the reference. Then the network is optimized by backpropagation.

7. **(2)** For comparison purposes, the forward pass and the loss are also calculated for the validation set within the same loop in each epoch. Insert the calculation for y_pred_val and loss_val.

In order to evaluate the training process we monitor how the loss changes with the number of epochs. Thereby both, train and validation loss is computed.
There is another figure that compares the predicted life expectancy with the reference. With a perfect model, these values are identical, i.e. all points are on the black line.

8. **(4)** Try to analyze the loss curves: How can you see from the loss curves how many epochs you should train a network? Please discuss the training loss as well as the validation loss. Therefore vary the hyperparameters learn_rate and num_epoch and look at the different loss curves.

9. **(5)** To further optimize the network, the architecture of the network can be improved: Change the net depth (vary number of hidden layer) and width (vary number of neurons in hidden layer) in the class RegressNet. Then briefly describe the influence on the loss curves and computing time with unchanged hyperparameters.
Hint 1: PyTorch documentation (https://pytorch.org/docs/stable/nn.html) is useful for more information about the layers.
Hint 2: Afterwards the hyperparameters should be adjusted to fit the new architecture.

10. **(5)** Improve the model by predicting life expectancy not only as a function of BMI, but of multiple features. Use any number of other features combined as X. To do this, you have to go back to the point where you split the data into X and Y. Do not use the first three columns, since they contain no float values or only meta data. Important: The input dimension of the network must also be adjusted. Note: If you make a change to an already executed cell in the .ipynb script, all other cells should also be executed again.

Finally, the trained neural network can be tested for new/unseen data.

11. **(3)** Insert the calculation of y_pred_test and loss_test. Look also at the mean absolute difference between the predicted and the reference life expectancy from the test set.

## Classification Network

12. **(15)** The step to the first classification network is not big anymore. For this task, you are provided with the source code **main_classification_iris._colab.ipynb**, which is very similar to that of the regression network. The iris-flower dataset consists of 160 iris, each described by 4 features and divided into 3 different classes. The aim is to use a simple neural classification network to predict the class from the features.

    - Solve the TODOs analogous to the regression network. Note that we now have to work with a one-hot encoded (https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179) target vector.
    - The evaluation is slightly modified: Along with the loss plots we use a confusion matrix now to compare reference and prediction.
    - It may be that the classification will not yet work optimally. This is because there is no activation function above the output of the network. Explain, why it makes sense to use a softmax activation for a classification task. Build such an activation with F.softmax(output, dim=1) into the architecture and compare the y_pred after the respective runs with and without activation.
      Note: Instead of manually adding a softmax function, you can also use another loss function e.g. CrossEntropyLoss(), which applies the softmax activation internally. We will use this in the following exercise.