Deep Learning KU (DAT.C302UF), WS25
Assignment 2
Training Simple Neural Networks

Simon Hitzginger
simon.hitzginger@tugraz.at

Teaching Assistant:	Leon Tiefenböck
Points to achieve:	25 pts
Deadline:	03.12.2025 23:59
Hand-in procedure:	You can work in groups of **at most two people**. **Exactly one** team member uploads two files to TeachCenter: **The report (.pdf)** and **the Python code (.py or .ipynb)**. The first page of the report must be the **cover letter**. Do not upload a folder. Do not zip the files.
Plagiarism:	If detected, 0 points for all parties involved. If this happens twice, we will grade the group with "Ungültig aufgrund von Täuschung"

We will use PyTorch to train a neural network for a regression task on the Concrete Strength Dataset. Our goal is to predict the strength of concrete based on various variables. I recommend that you set up a Conda environment and install the dependencies using `conda env create -f environment.yml -n deeplearningA2`. Your submitted code should not use any external libraries that are not included in this file.

Download the dataset from the TeachCenter and place it in the same folder as your Python file / Jupyter notebook. You can then load the data with:

```
df = pd.read_csv("data.csv")
```

There are a total of 1030 samples in the dataset. Table 1 shows an explanation of all variables, for further information you can also have a look at the Dataset Source.

Variable	**Unit**	**Description**
Cement	kg/m^3	Amount of cement in the mixture.
Blast Furnace Slag	kg/m^3	Amount of blast furnace slag in the mixture.
Fly Ash	kg/m^3	Amount of fly ash in the mixture.
Water	kg/m^3	Amount of water in the mixture.
Superplasticizer	kg/m^3	Amount of superplasticizer in the mixture.
Coarse Aggregate	kg/m^3	Amount of coarse aggregate in the mixture.
Fine Aggregate	kg/m^3	Amount of fine aggregate in the mixture.
Age	Days (1–365)	Age of the concrete when tested.
Strength	MPa	Compressive strength of the concrete.

Table 1: Concrete Mixture Components and Their Descriptions

The **target** variable will be the **Strength**. The other 8 variables will be the input features.

For all the tasks below, you may use the provided `.ipynb` **template** as a starting point and adjust the code as needed. Feel free to experiment and explore different approaches, but make sure that every step and design choice is clearly documented in your report so that your results can be fully reproduced. All relevant results, plots, and tables should be presented and discussed in the PDF report.

Task 1 – Data Preparation [3 Points]

In this task, you will prepare the provided dataset for machine learning experiments.

1. **Data Inspection and Splitting:** Make sure you loaded the dataset correctly. Split into train-, validation- and test-set with 60/20/20 % of the samples respectively. Also split the input features and target variable. Briefly explain the purpose of the three different sets. *Hint: you can use* `train_test_split` *from* `sklearn.model_selection`.
2. **Feature Visualization:** Visualize the distributions of all input features using suitable plots (e.g. histograms or boxplots). Comment briefly on observed characteristics, such as skewness and scale differences.
3. **Feature Scaling:** Standardize the input features, and explain why this is a good idea for this task. *Hint: use* `StandardScaler` *from* `scikit-learn`, *but make sure to fit the scaler only on the training set before applying it to validation and test data.*
4. **DataLoader Construction:** Convert the processed arrays into PyTorch tensors and create `DataLoader` objects for each data split. Choose and justify parameters such as batch size and whether to shuffle the data. These can stay the same for the rest of the assignment. If you want to change them for some reason, clearly state this in your report. *Hint: use* `torch.utils.data.TensorDataset`.

Task 2 – Feed-Forward Neural Network [7 Points]

In this task, you will design and train feed-forward neural networks for the regression task, and experiment with how different architecture choices affect training behaviour and model performance.

1. **Model Design and Training Setup:** Implement a feed-forward neural network architecture in PyTorch. Experiment with different numbers and sizes of hidden layers. Use ReLU as activation function for hidden units. Report and explain your choices for the output layer (number of neurons, activation function) and the loss function. Also define and justify your training setup, including optimizer, learning rate, number of epochs, and batch size. *Hint: you can use* `nn.Sequential` *or* `nn.ModuleList` *to dynamically adjust the layers of your network.*
2. **Training and Results Comparison:** Train each model using your defined setup. For each configuration, report the final train and validation loss as well as the best validation loss achieved during training. Summarize your results in a table comparing all models. *Hint: most of this is provided in the code template!*
3. **Parameter Count:** Compute and report the total number of trainable parameters for each model. Add this information to the table above. Verify this computation by adding up all the weights and biases of each layer for one of your models with at least two hidden layers.
4. **Learning Curves:** Select three different models (including the best one) and plot their training and validation losses across epochs. Comment briefly on the plots. Do you observe overfitting or underfitting in any of the models? Use both the plots and the result table to justify your answer.

Task 3 – Optimizers and Learning Rates [5 Points]

In this task, you will experiment with how the choice of the optimizer affects training behaviour and model performance, without changing the model itself.

1. **Optimizer and Training Setup:** Keep the network architecture fixed to the *best* model from Task 2. Report and justify your choice. You may choose a smaller model if training speed and parameter count justifies a slightly worse performance. With this model, vary the training setup by trying at least 3 different optimizers (like SGD, momentum SGD and Adam). Also try different hyperparameters for the optimizers (like learning rate or momentum). You may also adjust other related hyperparameters such as the number of epochs if needed. For each configuration, report the final train and validation loss as well as the best validation loss achieved during training. Summarize the results in a table.

2. **Learning Curves and Discussion:** Plot the training and validation losses over epochs for the different optimizers. Comment on convergence speed, stability, and generalization behaviour. Discuss which setup performs best and why.

3. **Learning Rate Scheduling:** Using two different optimizers from above, now train the model with two different learning rate schedulers. Plot the learning rate over epochs to verify that the schedulers are configured correctly! Compare the performance of each optimizer with and without schedulers, and report the results in a table. Discuss whether scheduling improves convergence or final validation performance. *Hint: you will need to implement this yourself in the code template!*

Task 4 – Weight Investigation [4 Points]

1. **Setup and Initialization:** Choose a good training setup (architecture, optimizer, learning rate, etc.) based on your results from Task 2 and Task 3. Report your chosen configuration and briefly justify your choice. Plot the distribution of all weight parameters for an **untrained** model. How does PyTorch initialize weights by default? Check the official documentation and verify your findings using your plots.

2. **Trained Model Weights:** Train the model with the chosen setup and visualize the learned weights as heatmaps for all layers. Comment on any visible structure or patterns. Can you easily make any observations about the influence of individual input variables on the target? *Hint: you can use the plotting functions from the template, but make sure the shapes are correct.*

Task 5 – Gradient Investigation [4 Points]

1. **Gradients for the Current Setup:** Choose a fixed training setup (architecture, optimizer, learning rate, epochs, batch size) based on your results from Task 3 and justify your choice briefly. Compute the average gradient magnitude for each layer according to

$$\bar{g}_\ell^{(e)} = \frac{1}{N_e} \sum_{i=1}^{N_e} \frac{1}{n_\ell} \sum_{j=1}^{n_\ell} \frac{1}{|\Theta_{\ell,j}|} \sum_{\theta \in \Theta_{\ell,j}} \left| \nabla_\theta \, \mathcal{L}(x_i, y_i) \right|,$$

   where N_e is the number of samples seen in epoch e, n_ℓ is the number of neurons in layer ℓ, and $\Theta_{\ell,j}$ are the parameters (including bias) of neuron j in layer ℓ. Plot the average gradient for each layer in the last training epoch and discuss what you observe. *Hint: use the template code!*

2. **Increasing Depth and Vanishing Gradients:** For this task, use a small number of neurons per hidden layer (e.g. 32) and gradually increase the number of hidden layers. For each configuration, compute and plot the per-layer average gradients to investigate how far the gradient propagates through the network. Identify and describe the point where gradients start to vanish. Discuss how this affects the training dynamics, and illustrate it by showing a scatterplot of model predictions versus targets for a configuration where vanishing gradients occur. *Hint: don't be shy, deep neural networks can have up to hundreds of hidden layers.*

Task 6 – Final Evaluation [2 Points]

Select the best hyperparameter settings based on the results of the previous tasks. Clearly summarize and justify your choices. Perform a final training run with this model on the entire training data (i.e., combine the original training and validation sets) and evaluate it on the test set.

Provide a scatter plot comparing the model predictions (x-axis) with the ground-truth target values (y-axis) for the test set. Report the final test loss and briefly comment on both the numerical result and the pattern observed in the scatter plot. What does the plot reveal about the model's predictive performance?