Exercise: 1

Deadline: 19 May 2024 23:59:59 GMT+1

Task 1: Sequence of the following exercises

The goal of this and the next exercise sheet, is the implementation of your own Deep Learning Framework in Python. In addition to the Python standard library, you may use exclusively *Numpy*, as well as a library of your choice that is only used to download the MNIST dataset.

Exam bonus You can work in groups of up to three students to receive a bonus for the exam. Here are the necessary steps:

(a) Contact Kirill Djebko via email (kirill.djebko@uni-wuerzburg.de) to obtain a GitLab repository.

(b) Upload your code for *each* subtask (e.g., 2 b) at least once to the repository.

(c) Make sure you have implemented all tasks of the first two exercise sheets in a functional manner. This includes that the framework is able to perform training on the MNIST dataset in a reasonable time.

(d) Make sure to upload the code for each exercise sheet to the repository in a timely manner. When you are finished with the sheet, create a *Tag* from the *Commit* that marks the end of your programming work for the exercise sheet. The commit must be uploaded *before the deadline*.

(e) Create a short guide as README.md in the repository on how to use your framework. If you use external libraries to download the MNIST data, also include a requirements.txt file including the name of the library.

(f) In the WueCampus course you can find tests, which each implemented layer must pass. You can modify the structure of the tests to fit your implementation but you must keep the test values.

(g) To receive the bonus, you as a group must be able to explain your code after completing the programming work. This will be done through a conversation in which you explain the functionality of your code and the decisions you made during the implementation. The scheduling procedure will be explained in advance.

Note: First, implement the functionality and then optimize your code with regard to runtime. Whenever possible, use slicing instead of loops

Task 2: Your own small deep learning framework

(a) Model the data as follows:

1. A tensor class in which the **elements**, the **deltas**, and a **shape** are stored.
2. A **shape** class that specifies the dimension of your data.

(b) Model your network as follows:

1. An **input layer** transforms the input data into tensors that can be read by the network.
2. Several **layers** process the data according to your definition.
3. One or more loss functions that evaluate the result of the last tensor.
4. The network should have methods **backprop()** and **forward()** to send input data through the network.

(c) Model your layers with at least the following methods:

- **forward()**
- **backward()**
- **calculate_delta_weights()**, if there are parameters in the **layer**.

(d) Layers to implement:

- Fully connected
- Activation (Relu, Sigmoid, TanH)
- Softmax

(e) Losses to implement:

- Cross-entropy
- Mean-squared error

(f) Optimizer

Implement the standard stochastic gradient descent as shown in the lecture.

(g) Weight initialization

Initialize all weights randomly in the interval [-0.5, 0.5].

(h) Saving and loading

Implement functions to save and load the weights of your network. Extend your network class with the methods “save_params(self, folder_path)” and “load_params(self, folder_path)”, which receive a path to a folder to save and load the weights, respectively. When saving, the weights of all layers with parameters should be stored in this folder (e.g., as .pkl files using the Pickle Python module). When loading, the weights should be loaded correctly. For this purpose, you can for simplicity assign attributes “layer_type” and “num” to each layer, from which a unique identifier for the layer can be derived, which you can use as a filename for the saved weights. Use “os.path.join()” for concatenating paths. You can save the weights and biases of each layer as separate files for simplicity. When calling the training method, it should be possible to select using a boolean whether the network should be trained (and in this case the new weights saved) or whether existing weights should be loaded instead.

Task 3: MNIST

(a) Test your implementation on the MNIST dataset. Let your implementation run for 10-20 epochs and note down for each epoch, the epoch no., the runtime for the epoch and the average loss for the epoch. Also note down the accuracy of the trained network on the test data. Your implementation should achieve an accuracy of $> 95\%$ (however, this is not a strict criterion but only serves as an aid). What observations do you make? Write down your notes and observations in the README.md file and commit them together with the saved network weights to the GitLab repository.