Neural network training

Solve the following exercises and upload your solutions to Moodle (unless specified otherwise) until the specified due date. Make sure to use the *exact filenames* that are specified for each individual exercise. Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are allowed to write additional functions, classes, etc. to improve readability and code quality.

Exercise 1 – Submission: `a5_ex1.py` 80 Points

Write a function

```
training_loop(
    network: torch.nn.Module,
    train_data: torch.utils.data.Dataset,
    eval_data: torch.utils.data.Dataset,
    num_epochs: int,
    show_progress: bool = False
) -> tuple[list, list]:
```

that provides the main functionality of training the `SimpleNetwork` from Assignment 4. To train the network on a regression task, a file `dataset.py` with the function `get_dataset()` is provided that returns a tuple of a training and evaluation `Dataset` of 32D samples with 1D target values.

The function `training_loop` should train and evaluate an object of the `SimpleNetwork` class on the datasets obtained by `get_dataset` with the following functionality:

- Create an optimizer of your choice that is responsible for updating the weights of your network.
- Create `DataLoader`s for `train_data` and `eval_data` with settings of your choice.
- Perform `num_epoch` full iterations over `train_data`. For every minibatch of each epoch:
  - Compute the mean-squared error loss of the given batch.
  - Update the network's weights according to this minibatch loss.
  - Collect the minibatch loss to compute the average loss of the epoch (averaged over all minibatch losses).
- After every epoch of training, a full iteration over `eval_data` is performed. Again, the loss needs to be computed and stored, but no weights should be updated.
- After training and evaluation, the function returns a 2-tuple, where the first entry is the list of (averaged) training losses and the second entry is the list of evaluation losses.
- If `show_progress` is set to `True`, progress bars should be shown during training (use `tqdm`).

Example program execution:

```
if __name__ == "__main__":
    from a4_ex1 import SimpleNetwork
    from dataset import get_dataset

    torch.random.manual_seed(0)
    train_data, eval_data = get_dataset()
    network = SimpleNetwork(32, 128, 1)
    train_losses, eval_losses = training_loop(network, train_data, eval_data, num_epochs=10)
    for epoch, (tl, el) in enumerate(zip(train_losses, eval_losses)):
        print(f"Epoch: {epoch} --- Train loss: {tl:7.2f} --- Eval loss: {el:7.2f}")
```

Example output (might vary due to implementation differences):

```
Epoch: 0 --- Train loss: 1590.81 --- Eval loss: 1419.72
Epoch: 1 --- Train loss: 1267.04 --- Eval loss: 1242.50
Epoch: 2 --- Train loss: 1071.32 --- Eval loss: 1056.60
Epoch: 3 --- Train loss:  949.88 --- Eval loss:  821.91
Epoch: 4 --- Train loss:  786.39 --- Eval loss:  671.86
Epoch: 5 --- Train loss:  648.59 --- Eval loss:  649.52
Epoch: 6 --- Train loss:  523.82 --- Eval loss:  435.60
Epoch: 7 --- Train loss:  382.02 --- Eval loss:  327.81
Epoch: 8 --- Train loss:  355.79 --- Eval loss:  305.20
Epoch: 9 --- Train loss:  282.25 --- Eval loss:  476.98
```

Hints:

- Do not forget to set your network to training/evaluation mode before training/evaluating it.
- If you do not see fast decrement of the training loss, you likely have an error in your code or your learning rate is not set appropriately.
- When testing your function and creating a `SimpleNetwork`, the parameters `input_neurons` and `output_neurons` need to be set to 32 and 1, respectively. `hidden_neurons` can be arbitrary.

Exercise 2 – Submission: `a5_ex2.py` 20 Points

Update the function `training_loop` from Exercise 1 to include *early stopping*.

- Early stopping should prevent overfitting to the training data by stopping training when no improvements on the evaluation dataset are made. Add this functionality to `training_loop` by finishing/leaving the training loop when no new minimal evaluation loss was achieved in the last 3 epochs.
- The function should still return the 2-tuple with the training and evaluation loss lists when training is finished (either by reaching `num_epochs` epochs or by early stopping).

Additionally, create the function `plot_losses(train_losses: list, eval_losses: list)` to plot the training curves for the two losses after training is finished. The plot must contain the following (all other settings are completely up to you):

- The x-axis should show the epochs.
- The y-axis should show the mean-squared error loss.
- There should be two curves in different colors for the the training and evaluation losses.
- Add a legend to distinguish both curves.

Example program execution:

```
if __name__ == "__main__":
    from a4_ex1 import SimpleNetwork
    from dataset import get_dataset

    torch.random.manual_seed(0)
    train_data, eval_data = get_dataset()
    network = SimpleNetwork(32, 128, 1)
    train_losses, eval_losses = training_loop(network, train_data, eval_data, num_epochs=100)
    plot_losses(train_losses, eval_losses)
```

Example output plot (might vary due to implementation differences):

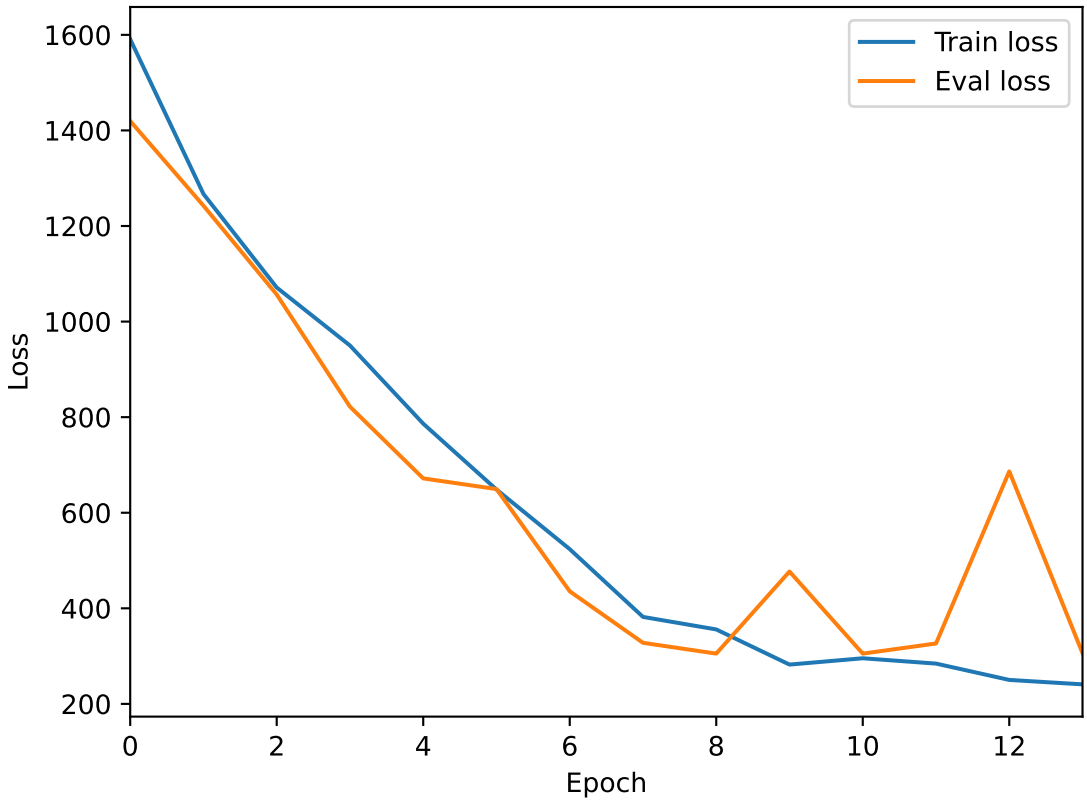

Figure 1: Example plot showing the training and evaluation losses. Here, early stopping was applied after 13 epochs (instead of the 100 epochs that were initially specified, see example code above).

Hints:

- While we would ideally also store the model with the lowest evaluation loss, this is *not* required in this exercise. Just leaving the training loop (and thus using the updated model with the higher evaluation loss) is enough.
- Use matplotlib for plotting.