Deep Learning
7th Assignment

F. Hutter & A. Valada
WS 2023/2024

To get access to this week's code use the following link: https://classroom.github.com/a/wyda8yqG

General constraints for submissions: Please adhere to these rules to make our and your life easier! We will deduct points if you fail to do so.

- Your code should work with *Python 3.8*.
- You should only fill out the *TODO-gaps* and not change anything else in the code.
- Add comments to your code, to help us understand your solution.
- Your code should adhere to the PEP8 style guide. We allow line lengths of up to 120.
- While working on the exercise, push all commits to the `dev` branch (details in assignment 1). Only push your final results to the `main` branch, where they will be automatically tested in the cloud. If you push to `main` more than 3 times per exercise, we will deduct points.
- All provided unit tests have to pass: In your *GitHub* repository navigate to *Actions* → your last commit → Autograding → education/autograding to see which tests have passed. The points in autograding only show the number of tests passed and have nothing to do with the points you get for the exercise.
- `for` loops can be slow in Python, use vectorized `numpy` operations wherever possible (see assignment 1 for an example).
- Submit *a single PDF* named `submission.pdf`. Include your matriculation numbers on the top of the sheet. Add the answers and solution paths to all non-coding questions in the exercise. Do not leave answers to any questions as comments in the code. You can use Latex with the student template (provided in exercise 1 / ILIAS) or do it by hand.
- Please help us to improve the exercises by filling out and submitting the `feedback.md` file.
- We do not tolerate plagiarism. If you copy from other teams or the internet, you will get 0 points. Further action will be taken against repeat offenders!
- Passing the exercises ($\geq 50\%$ in total) is a requirement for passing the course.

How to run the exercise and tests

- See the `setup.pdf` in exercise 1 / ILIAS for installation details.
- We always assume you run commands in the *root folder* of the exercise repository.
- If you use miniconda, do not forget to activate your environment with `conda activate mydlenv`
- Install the required packages with `pip install -r requirements.txt`
- Python files in the *root folder* of the repository contain the scripts to run the code.
- Python files in the `tests/` folder of the repository contain the tests that will be used to check your solution.
- Test everything at once with `python -m pytest`
- Run a single test with `python -m tests.test_something` (replace `something` with the test's name).
- To check your solution for the correct code style, run `pycodestyle --max-line-length 120 .`
- The scripts `runtests.sh` (Linux/Mac) or `runtests.bat` (Windows) can be used to run all the tests described above. If you are on Linux, you need execution rights to run `runtests.sh`.

This exercise focuses on recurrent neural networks (RNNs). Also, we will be using PyTorch[1] from now on, more specifically the CPU variant of version 1.7.1.

- Install PyTorch
- Implement an LSTM cell in PyTorch
- Train two forms of RNNs, namely:
  - Many-to-one: value memorization
  - Many-to-many: de-noising sequences

1. Warmup: PyTorch installation

PyTorch is a mature python library for deep learning. It provides modules like the ones we implemented in the previous exercises and many more. We tried to keep the interface similar, so you will already feel at

[1] If you would like to have a brief introduction to PyTorch you can have a look at the Introduction to PyTorch or the Deep Learning with PyTorch: A 60 Minute Blitz tutorials.

home when using PyTorch. In contrast to our own implementation, when using PyTorch you only have to implement the forward pass - the backward pass will be calculated automatically with autograd.

Internally, PyTorch's `autograd` creates a Directed Acyclic Graph (DAG) of all the tensors and all operations performed on those tensors. Those operations are objects that know how to calculate their forward pass, but also their derivative for the backpropagation. In the constructed DAG the input tensors are the leaves, and the output tensors are the root. `autograd` uses this DAG to apply the chain rule and calculate the gradients. The backpropagation eventually begins when `.backward()` is called on the root of the DAG.

In this exercise and the following exercises, PyTorch will be installed automatically via `requirements.txt`.

Todo: Perform the following step to verify that PyTorch is installed correctly:

1) **CPU only:**

   After installing the requirements, run the file `run_cpu_test.py` to test your PyTorch installation.

   In this exercise, we do not use CUDA since training small models with little data is fast enough on CPU.

2) **GPU support:** Training on GPUs is a lot faster than training on CPUs. Make sure your GPU drivers are updated to the latest version. GPU version has to be installed manually. Please be sure to remove the commands for PyTorch installation from `requirements.txt` if you want to install the GPU variant.

   More specifically remove the two following lines:

   - `-f https://download.pytorch.org/whl/torch_stable.html`
   - `torch==1.7.1+cpu`

   Quick installation: Install PyTorch with the latest cuda version, run file `run_gpu_test.py` and if it crashes, uninstall PyTorch and install it again with the next older version of cuda.

   Long installation: Check CUDA-Enabled GeForce and TITAN Products. Note the CUDA compute capability of your GPU and check for the correct CUDA Version for your device.

   Tips / Troubleshooting CUDA: Whenever you want to use CUDA, remember to move both the model and the data to GPU. If you forget to move something, you will see errors like `Expected object of device type cuda but got device type cpu` or `expected torch.cuda.FloatTensor but got torch.FloatTensor`.

   If you get `CUDA out of memory`, the causes could be one of the following:

   The computation does not fit on your GPU memory. Reduce the batch size, reduce the model size, use the CPU or buy a new GPU with more memory.

   There may be other processes running that use up GPU memory. For example, if you use Jupyter notebooks, they stay open in the background even when you close the browser window. Go to the Jupyter main page, choose the "Running" tab and shutdown all existing notebooks. Run `nvidia-smi` in the command line to see which processes block your memory.

2. Preliminary Questions

Before we dive into the implementation parts, let's think about a few aspects of recurrent neural networks. *Hint: Reading chapter 10 of the deep learning book might help with answering the questions.* Two other great resources are:

- Andrej Karpathy's blog post on RNNs, which, among other things, nicely explains the different variants of RNNs (one-to-one, many-to-many, etc).
- Chris Olah's blog post on LSTMs.

1) [2 points] (core) In what sense are convolutional neural networks and recurrent neural networks similar? In what sense are they different?

2) [2 points] (core) How can one counteract vanishing or exploding gradients in RNNs, allowing to learn long-term dependencies?

3. Coding Tasks

1) [5 points] **LSTM cell**

In PyTorch, all layers inherit from `nn.Module` and implement the forward function (the backward pass is computed automatically). Parameters should be initialized in the constructor.

To get a feeling for how layers are implemented in PyTorch, you can for example take a look at the source code of the Linear layer ($h = wX + b$).

Your task here is to implement the LSTMCell, which takes a feature tensor and the hidden state as input and returns the new hidden state (sometimes also referred to as output) and the new cell state.

Todo: [2.5 points] Fill the TODOs in `LSTMCell` (`lib/models.py`)

Todo: [2 points] Fill the TODOs in `LSTM` (`lib/models.py`)

As a simple test, we will see if the LSTM can learn to echo a value at a specific index of the sequence. If your implementation is correct, you should get more than 90% accuracy.

Todo: [0.5 points] Run `train_lstm_echo.py` to train the LSTM model for 100 epochs.

2) [5 points] (core) **LSTM Use Case - Noise Removal** Implement an RNN to remove noise from different sine function instances. If you didn't finish the `LSTM` implementation part, you can use `nn.LSTM` here.

The goal is to remove Gaussian noise from a sequence generated from a sine function. To get an idea what the data looks like, plot six different sine function instances with and without noise.

Todo: [1 point] Complete function `plot_functions_with_noise` (`lib/plot.py`). Run `python plot_noise.py` to see your plots.

Now it's time to define the model! Let's stack two LSTMs (not LSTMCell):

- The first with input shape (batch_size, sequence_length, input_size) and with output shape (batch_size, sequence_length, hidden_size)
- The second with input shape (batch_size, sequence_length, hidden_size) and with output shape (batch_size, sequence_length, hidden_size)
- followed by a Linear layer which takes a (batch_size, sequence_length, hidden_size) vector as input and outputs a tensor with shape (batch_size, sequence_length, 1).

Todo: [1.5 points] Fill the TODOs in `NoiseRemovalModel` (`lib/models.py`). The output of the first LSTM should flow as input into the second LSTM. To allow the model to see some values before estimating the output, pad the sequence accordingly.

Todo: [1.5 points] Fill the TODOs in function `train_noise_removal_model` (`lib/experiments.py`) and then run `train_noise_removal.py`. Around 50% of noise should get removed after training.

Todo: [1 point] Which design pattern from the ones mentioned in the lecture is used for the Noise Removal Task and why? What is the shape of the input tensor and how does it change as we propagate through the two LSTM layers and the final linear layer?

3) [2 points] **Hyperparameter Optimization** Play with the model's hyperparameters and try to improve the amount of noise removed. List at least three different configurations you have tried and the respective percentage of noise removed. Make sure to always create a new model and that you train and validate on the same data! Later in this course we will see methods for automating the search of good hyperparameters.

A few possibilities:

Configuration	Noise removed in percent
Initial configuration	49.8%
Train for 200 epochs	55.8%
Hidden size 20	47.6%
Hidden size 60	53.2%
Hidden size 40, shift 3	51.5%

Todo: [2 points] Complete function `run_hpo` (`lib/experiments.py`) and run `run_hpo_experiments.py`

4. [1 bonus point] **Code Style**

On every exercise sheet, we will also make use of `pycodestyle` to adhere to a common python standard. Your code will be automatically evaluated on submission (on push to master). Run `pycodestyle --max-line-length=120 .` to check your code locally.

5. [1 bonus point] **Feedback**

Todo: Please give us feedback by filling out the `feedback.md` file.

- Major Problems?
- Helpful?
- Duration (hours)? For this, please follow the instructions in the `feedback.md` file.
- Other feedback?

This assignment is due on 12.12.2023 23:59 CET. Submit your solution for the tasks by uploading (`git push`) the PDF, txt file(s) and your code to your group's repository. The PDF has to include the name of the submitter(s). Teams of at most 3 students are allowed.