Deep Learning (FSS24)
Assignment 2: RNNs

University of Mannheim

The archive provided to you contains this assignment description, datasets, as well as Python code fragments for you to complete. Comments and documentation in the code provide further information. Please note the following:

- It **suffices to fill out the "holes" that are marked in the code fragments** provided to you, but feel free to modify the code to your liking. You need to stick with Python though.
- Please **adhere to the following guidelines** in all the assignments. If you do not follow those guidelines, we may grade your solution as a FAIL. Provide a single ZIP archive with name `dl24-<assignment number>-<your ILIAS login>.zip`. The archive needs to contain:
  - A **single PDF report** that contains answers to the tasks specified in the assignment, including helpful figures and a high-level description of your approach. **Do not simply convert your Jupyter notebook to a PDF!** Write a separate document, stay focused and brief. You must **stay below 10 pages**, excluding references and appendix.
  - All **the code that you created** and used in its original format.
  - A **PDF document that renders your Jupyter notebook with all figures.** (If you don't use Jupyter, then you obviously do not need to provide this.)
- **A high quality report is required to achieve an EXCELLENT grade.** Such a report is self-explanatory (i.e. do not refer to your code except for implementation-only tasks), follows standard scientific practice (e.g. when using images, tables or citations), does not include hand-written notes, and does not exceed 10 pages. In addition, label all figures (and refer to figure labels in your write-up), include references if you used additional sources or material, and use the tasks numbers of the assignments as your section and subsection numbers.
- You **may work on this assignment in pairs**—i.e, with one (and only one) additional student—and then hand in a pair submission. To do so:
  - The PDF report and notebook must clearly report then **name and ILIAS login** of **both students** right at the beginning.
  - **Both students must submit** the same assignment on ILIAS separately.

  To be clear, if only one student of the pair submits the assignment, then only this student will receive the grade. Hand-in your solution via ILIAS until the date specified there. This is a hard deadline.

Assignment Description

In this assignment, you will perform and investigate basic sentiment analysis (positive/negative) for movie reviews (text) using PyTorch. The available data is provided to you as `reviews_small.txt` (input) and `labels_small.txt` (corresponding labels) in the `data` directory.

Note. This assignment requires the TorchText package. You may need to install this package on your system or update your Docker configuration.

Tasks 1–3 are pure implementation tasks; you do not need to discuss them in your report (but hand in your code, of course).

1 Datasets

PyTorch provides an abstract Dataset class to represent datasets, each consisting of a set of keys and corresponding data samples. Although PyTorch also provides a number of subclasses and default implementations for common cases, you are going to write your own dataset from scratch in this task.

We provide the stub class `ReviewsDataset` to get started. Here, each key corresponds to the item number (line number in raw file, 0-based) and data samples to the corresponding (movie review, label)-pair. Your task is to complete this class by ultimately implementing both the `__len__` and the `__getitem__` methods (see the Dataset documentation).

a) Preprocess the data. To do so, complete the `_preprocess_reviews` method (to tokenize the reviews and remove punctuation) and the `_preprocess_labels` (to convert labels to binary values).

b) Implement the `__len__` and `__getitem__` methods and test your code.

c) Observe that the provided code builds a vocabulary `self.vocab`, i.e., a bijective mapping from textual *tokens* to numeric *token ids*. Change your code so that `__getitem__` returns a list of token ids instead of a list of tokens when `use_vocab=True`. Test your code.

2 Data Loaders

To process data during training, PyTorch makes use of a `DataLoader`.

a) Familiarize yourself with the documentation and run the provided example `dataloader`. Now, modify the dataloader by increasing the batch size. You will notice that the dataloader now fails (why?).

b) PyTorch dataloaders accept a so-called `collate_fn` to perform modifications to each batch in the data loader (e.g., data augmentation). Here we will use the collate function to pad or trim all reviews to the same number of tokens:

   (i) Add zero padding (token `<pad>`), token id 0) for all reviews that are shorter than `MAX_SEQ_LEN`.

   (ii) Crop all reviews longer then `MAX_SEQ_LEN`.[1]

   Complete the `reviews_collate_fn` provided to you and construct data loaders for training, validation, and testing.

[1]Alternatively and more realistically, one may pad all reviews to the length of the longest review in the current batch (referred to as *dynamic batching*). However, this may increase the computational complexity and might require a GPU for training sufficiently fast, so we don't do this here.

3 Recurrent Neural Networks

We are now ready to train an RNN. In this assignment, we are going to use PyTorch's pre-canned layer implementations (see torch.nn). In particular, we'll use (in this order):

- Embedding layer (torch.nn.Embedding) to embed input tokens.
- LSTM encoder (torch.nn.LSTM) to produce a thought vector (= last-layer hidden state of last input token).[2]
- Linear layer with one hidden unit (torch.nn.Linear) to produce logit scores from the thought vector.
- Logistic function (torch.nn.Sigmoid) to produce the model's probability of a positive sentiment.

Consult the PyTorch documentation on how these modules are to be used. Pay close attention to the LSTM module and its *Inputs* section, which describes the required shape of the input sequence and the initial hidden states. We provide a stub implementation called `SimpleLSTM`.

a) Complete the `init_hidden` method.

b) Complete the constructor. To do so, instantiate all of the individual components given above in `__init__`.

c) Implement the `forward` pass. The method obtains a batch from the data loader as input, and should return a tuple consisting of:

   (i) A tensor of predicted probabilities (shape: `batch_size` $\times$ 1). This is the model output.

   (ii) A tensor of the thought vectors (shape: `batch_size` $\times$ `hidden_dim`). We will use this for visualization later on.

d) Complete the evaluation function `reviews_eval` to compute accuracy and average loss. We will use this function for validation during training and for final testing.

e) Consult the PyTorch documentation on how to properly implement an optimization loop. Then, complete the implementation of the training function `reviews_train`.

[2]We will use dropout to regularize the LSTM encoder. Simply set the `dropout` argument of the encoder accordingly (see provided code).

4 Pretrained Embeddings & Visualization

You are provided with GloVe word embeddings in the file `data/word-embeddings.txt`.[3]

a) Describe and explain the content of the file `data/word-embeddings.txt`.

b) Load the pretrained embeddings for the subset of the words used in this assignment into a plain embedding layer (code provided). Explore the embeddings using t-SNE and other methods of your choice and discuss your findings.

c) Train the provided `SimpleRNN` model (with the provided hyperparameter settings) for 10 epochs. Then use t-SNE to visualize

   (i) the embeddings of the thought vectors of the training dataset, and

   (ii) the embeddings of the thought vectors of the validation dataset.

   Discuss your findings.

d) Repeat the previous subtask, this time initializing the word embedding layer with the Glove embeddings (code provided; that's *pretraining with finetuning*). Discuss.

e) Repeat the previous subtask, this time additionally freezing the word embeddings to the Glove embeddings (code provided; that's *pretraining without finetuning*). Discuss.

5 Exploration

In this task, you will explore different hyperparameter settings for training as well as different RNN architectures. You can use the code that you created so far (and, in particular, the `SimpleLSTM` model) as a blueprint.

a) Starting from the model of task 4d (i.e., pretraining with finetuning), study individually the consequence of changing:

   - the dropout probability (at least: 0 and 0.8),
   - the choice of RNN cell (at least: Elman cells, GRU cells, LSTM cells),
   - whether the RNN is unidirectional or bidirectional.

   Your submission must include and discuss at least the above seven different setups. Report and discuss your results, use visualizations as suitable, and draw conclusions as to which design appears more suitable for the current problem.

b) Optimize the model (e.g., dimensionality or choices from above) and training process (e.g., dropout, early abort on validation). Once you are satisfied with your model's validation performance (and only then – recall golden rule of ML), evaluate on the test data. Document your process, results, and discuss.

Note. Take care of your code quality for this task. Make sure we are able to reproduce your findings from your notebook.

[3]This file contains a subset of the complete GloVe embeddings available at `http://nlp.stanford.edu/data/glove.6B.zip`