Deep Learning (FSS22)
Assignment 2: CNNs and RNNs

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.

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 `dl22-<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. As a guideline, try to stay below 10 pages.
- 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 on 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.

Hand-in your solution via ILIAS until the date specified there. This is a hard deadline.

Datasets

In this assignment, we explore two real-world tasks using convolutional neural networks (CNNs) and recurrent neural networks (RNNs) in PyTorch. The datasets provided in both cases are small so that you do not have to use substantial computational resources (even a CPU works out). In practice, larger datasets, considerably more involved models, and sufficient computational resources are often required to obtain good performance.

1 Convolutional Neural Networks

The goal of this task is to get familiar with using PyTorch for a real-world task. This includes loading data, constructing batches, defining and training a model, and evaluating model performance.

We work with the Fashion-MNIST dataset, which is a set of images of Zalando's clothing items. The dataset consist of a set of images of 28x28 pixels, each a gray-scale value between 0 and 1. Code to load the dataset is provided to you. Familiarize yourself with the dataset. For more information, see `https://github.com/zalandoresearch/fashion-mnist`

a) Consider a simple CNN with the following architecture:

   (i) 3x3 convolutional layer with bias, 1 input channel, 32 output channels, stride 1, padding 1 on each side (`torch.nn.Conv2d`)

   (ii) Logistic activation function (`torch.nn.Sigmoid`)

   (iii) 2x2 max-pooling layer with stride 2 (`torch.nn.MaxPool2d`)

   (iv) Linear layer with 10 hidden units (`torch.nn.Linear`)

   (v) Log-softmax function (`torch.nn.LogSoftmax`)

   State the shapes of the input and output volume of each component as well as the number of its parameters. How can we interpret the network output for a given example?

b) Implement the above CNN by completing the provided code fragment. First define the individual components in `SimpleCNN#__init__`, then use them in the computation of the forward pass in `SimpleCNN#forward`.

c) Complete the `mnist_test` function to test your model's performance on test data by computing the model *accuracy*. Here we make use of PyTorch's data loaders (`torch.utils.data.DataLoader`), which provide automatic batching, shuffling, and parallelization (batch creation can be a bottleneck in practice).

d) Complete the `mnist_train` function to train your CNN. Use what you have learned in the previous subtasks. Your implementation should construct batches and, for each batch, perform a forward pass, a backward pass, and an optimizer call (see example in the lecture). Use negative log-likehood as loss function (`torch.nn.NLLLoss`) and Adam (`torch.optim.Adam`) as optimizer.

e) Train your model for 5 epochs with a batch size of 100 and a learning rate of 0.001. How does the model perform on the task? Is the performance similar across classes? Why or why not? When it makes mistakes, are they reasonable? Discuss.

f) **Optional.** Explore different architectures. Consider adding additional layers—e.g., convolutional, fully connected, dropout (`torch.nn.Dropout`), batch normalization (`torch.nn.BatchNorm2d`—or regularization terms.

2 Recurrent Neural Networks and Pretraining

In this task, we explore the usefulness of pretrained models and transfer learning. We perform sentiment analysis of IMDB movie reviews using RNNs, i.e., we classify each review as either positive or negative. As in the previous task, we use a small subset of the available data. This subset is provided in the `data/` folder. The complete dataset can be found at `https://ai.stanford.edu/~amaas/data/sentiment/`.

We create an RNN with LSTM units and train it from scratch, with pretrained word embeddings but without finetuning, and with fine-tuning. For pretraining, we use GloVe word embeddings; see `https://nlp.stanford.edu/projects/glove/`.

Start by familiarizing yourself with the code, including the file `helper.py`, which contains several helper functions for this task.

a) Consider a simple RNN with the following architecture:

   (i) Embedding layer (`torch.nn.Embedding`)

   (ii) LSTM encoder to produce thought vector (`torch.nn.LSTM`)

   (iii) Dropout layer (`torch.nn.Dropout`)

   (iv) Linear layer with one hidden unit (`torch.nn.Linear`)

   (v) Sigmoid function (`torch.nn.Sigmoid`)

   The architecture is parameterized by the hyperparameters `vocab_size`, `embedding_dim`, `hidden_dim`, `lstm_dropout_prob`, and `dropout_prob`.

   Implement the above RNN by completing the provided code fragment. First define the individual components in `SimpleLSTM#__init__`, then use them in the computation of the forward pass in `SimpleLSTM#forward`.

b) Train and evaluate the model (code provided) with the provided hyperparameters. Did the model converge after 5 epochs? Why do you think the model converged or did not converge?

c) You are provided with GloVe word embeddings in the file `data/word-embeddings.txt`. [1] Describe the content of the file.

d) Initialize a model with the provided word embeddings. Train the model once without fine-tuning and once with fine-tuning the embeddings (5 epochs, same hyperparameters as above). Compare and discuss the results of both approaches.

   Hint. You can avoid updates on the embedding layer by setting the flag `requires_grad` to false on the `weight` attribute of the embedding layer.

e) **Optional.** Explore different architectures and hyperparameters.

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