Preview, Exercise 5 - **Assignment 2**

Deep Learning Lab
Due: Sunday 13 November 2022, 10:00 pm (time in Lugano)
(Note: this is one week later than the originally announced two-week deadline)

October 23, 2022

Submission Instruction

You should deliver the following by the deadline:

- **Report**: a single *pdf* file that clearly and concisely provides evidence that you have accomplished each of the tasks listed below. The report should not contain source code (not even snippets). Instead, if absolutely necessary, briefly mention which functions were used to accomplish a task. All figures should have a caption, and there should be a text that refers to it (see the Latex documentation if you are not familiar with this). Please also make sure that the section numbering in your report exactly follows that of the assignment.
- **Source code**: a single Python file (*.py*) that can be used to accomplish each of the tasks listed above. The source code will be read superficially and checked for plagiarism. Importantly, if a task is accomplished in the code, but not documented in the report, it will be considered missing. Therefore, please make sure to report everything in the report. Note: Jupyter/Colab notebook files (*.ipynb*) are not accepted.

Please carefully read the instructions above to prepare your submission. Failure to stick to these rules may result in reduction of points. As stressed in class, it is strictly forbidden to exchange your code with others or copy texts/code from the internet. The score of 0 will be given to all students involved in such cases.

NB: You should use a GPU for this assignment.

1 Image Classification Using ConvNets [90/100 points]

The CIFAR-10 dataset contains $50,000$ training images and $10,000$ test images. Each image is of size 32×32 with three channels for red, green, and blue. Each image belongs to one of ten classes: plane, car, bird, cat, deer, dog, frog, horse, ship, or truck. Your task is to implement a convolutional neural network to correctly classify the images.

1.1 Dataset

Like the MNIST dataset we used in Exercise 4, the CIFAR-10 dataset is already available in PyTorch via `torchvision.datasets.CIFAR10`. The corresponding API documentation can be found here.

1. (4 pt) Load the corresponding dataset. Check the number of images. You can plot images using `matplotlib.pyplot.imshow`. **Visualize** (at least) four images from the training set. Be careful with the shape expected by `imshow`: the color channel needs to be the last dimension.
2. (10 pt) While `torchvision.datasets.CIFAR10` provides images with values taking between 0 and 1 by default, these values are not standardized to be zero-mean and unit-variance. Instead

of feeding these images as-is to the model, we can first normalize these values for each channel. The `transform` argument of `torchvision.datasets.CIFAR10` allows to apply multiple transformations to the original image data. For that, multiple transformations can be first composed via `torchvision.transforms.Compose`, and the result can be given as an argument to `transform`. Use `transforms.Normalize` to normalize the data to have mean 0 and standard deviation (std) 1 (carefully check its documentation). *Hint: means/stds for each channel in the original training data are: means=(0.4914, 0.4822, 0.4465) and stds=(0.247, 0.243, 0.261). You can use the same values for all data splits: training, validation (introduced later), test.*

3. (Bonus, 10 pt) Write and execute code to verify that the mean/std statistics provided above computed using all $50,000$ training images (which includes the validation part introduced in the next question) are correct.

4. (5 pt) The original dataset does not contain a validation set; we need to create one. We'll follow the common procedure (that we saw in Exercise 4): Split the original training set into a validation set (the last $1,000$ images) and an effective training set (the first $49,000$ images). We remind you that you can specify a sampler to be used in a dataloader (`torch.utils.data.DataLoader`) by providing the `sampler` argument. In our case, the SubsetRandomSampler is useful because it will pick samples from a given list of indices. Create two separate dataloaders for the training and validation set by using the same original dataset but with different subset samplers.

1.2 Model

1. (10 pt) Implement the following convolutional neural network using rectified linear activation functions (ReLUs):

   (a) Convolutional layer 1: 32 filters, 3×3.

   (b) Convolutional layer 2: 32 filters, 3×3.

   (c) Max-pooling layer 1: 2×2 windows.

   (d) Convolutional layer 3: 64 filters, 3×3.

   (e) Convolutional layer 4: 64 filters, 3×3.

   (f) Max-pooling layer 2: 2×2 windows.

   (g) Fully connected layer 1: 512 units.

   (h) Output layer.

   For hyper-parameters that are not specified above (e.g., padding), you can use the default parameters of the corresponding PyTorch implementations. *Hint: Be careful with the activation function for the output layer.*

1.3 Training

1. (20 pt) Implement the training pipeline. Make sure that you can monitor the current training loss and accuracy every n steps (with a reasonable choice of n), as well as, validation accuracy after each epoch. You should also keep track of the **best validation accuracy** and the corresponding epoch.

2. (1 pt) Use the following *hyperparameters* to train your model:

   (a) SGD optimizer with learning rate 10^{-3} and momentum 0.9.

   (b) A batch size of 32.

   (c) Train for a total of 20 epochs.

   (d) Use the cross entropy loss.

3. (5 pt) Train your model. The final validation accuracy should be above 70%.

4. (10 pt) **Plot** the evolution of the loss and accuracy for your training and validation set. Compare the two plots (train vs. validation) and **comment**.

5. (20 pt) There are many ways to improve performance. Besides searching for better hyperparameters, we may apply model regularization and data augmentation. For the simple architecture described above, dropout is an appropriate way to regularize. Study how to use dropout (`torch.nn.Dropout`) in PyTorch, and add a dropout layer after each max-pooling layer (for this question, do not put it anywhere else!). **Explain** what the argument `p` is. Try three different values of `p`, {0.2, 0.5, 0.8}, **plot** the learning curves (similar to Q4 above) for each run, and comment how they differ from the previous run without dropout. Note, dropout might slow down convergence; increase the number of epochs if necessary (there is no need to change other hyper-parameters). Your best model should achieve a validation accuracy over 75%.

6. (10 pt) **Report** the test set accuracy of the best model you obtained above. Randomly select (at least) 4 validation images, **visualize** them and **report** the output probability distributions of your best model for each output class for each image. *Hint: the order of the labels is* `('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')`.
Are the model predictions reasonable/correct/wrong?

7. (Bonus, 10 pt) Achieve a test accuracy above 80%. Describe all changes you needed to achieve it. Plot the learning curves and report the final test performance. *NB: This is an "open-ended" question for those who are curious.* You should try any ideas/parameters that come to your mind to improve performance. For example, last year, some students replaced the ReLU activation fuction by GELU, or added more layers, ... etc. You can report anything you have tried, but the bonus point will be given only if you effectively achieve a test accuracy above 80%.

1.4 Questions [5/100 points]

Provide a **brief** answer to the following questions **in your own words** and add them to your report. You can find more information in the relevant subsections of chapter 6,8,9 of the Deep Learning Book.

1. (1 pt) When (for what purpose) do you use softmax activation? (max one sentence)

2. (1 pt) Here we used a non-zero momentum in the SGD optimizer. In this case, the optimizer accumulates a *certain quantity* with a decay factor specified by the momentum value. Name this quantity. *Hint: read the PyTorch documentation of* `torch.optim.SGD`.

3. (1 pt) Here we used 2D convolution for images. Name one example type of data for which you may want to use 1D convolution.

4. (2 pt) Test time behavior of dropout is different from its behavior during training. Describe both behaviors (max two sentences). *Hint: you can read the PyTorch documentation of* `nn.Dropout`.