Assignment 1: Playing with PyTorch

CPSC 532R/533R Visual AI
by Helge Rhodin and Yuchi Zhang

This assignment introduces you to PyTorch and the training of deep learning models for visual computing. It will form the basis for future assignments and, most importantly, your course project. At the end of this assignment, you will be able to load training data, construct a neural network, train the network, and evaluate its accuracy; all in PyTorch.

PyTorch is a very flexible deep learning framework that allows you to write dedicated training code and custom visualization functions, yet, interfaces with high-level tools, such as tensorboard. We neither want to enforce a particular coding style nor make you reinvent the wheel. Hence, we let you play with an existing tutorial/implementation of your choice.

This assignment is split into several tasks. To make grading possible, augment your code with a comment `# Task N` that precedes your changes for task N. For instance for Task 2 your code might look like

```
# Task II
def __getitem__(self, dictionary):
    ...
```

Setup python 3.7, PyTorch (version 1.3.1) and jupyter lab (cuda is optional if you have a GPU). You have the following options (you can pick but we recommend trying both):

a) Install anaconda, jupyter lab (through anaconda navigator), and PyTorch (through conda command) on your laptop/PC. For instance, by following
docs.anaconda.com/anaconda/navigator,
towardsdatascience.com/jupyter-lab..., and
pytorch.org/get-started....

b) Register for Google Colab (free of charge, provides a jupyter notebook, PyTorch already installed).
colab.research.google.com

Pick a problem (dataset and task) that involved reconstructing from images (computer vision) or the generation of images (computer graphics) with a neural network. Such as classifying images, segmenting salient objects in images, reconstructing 3D pose, or generating images or shapes.

For students unfamiliar with PyTorch, we suggest using the jupyter notebook introduced in the lecture. Its simplicity makes it easier to explore PyTorch's core functionality. Those familiar with PyTorch may start from any tutorial or codebase, including your past projects should they apply. For instance, the following tutorial introduces tensorboard for interactive display of training progress https://towardsdatascience.com/build-a... and associated colab file https://colab.research.google.com/...

- Your starting code must include the training of a network (e.g., some variants of style transfer don't and should not be addressed here).
- If you have limited compute power (on your laptop), choose a problem that has small images (28x28 resolution) and can be solved with a small (shallow network).

- You must use a jupyter notebook as entry point (may use additional .py files) for starting the training and evaluation. Rename that notebook to `assignment1.ipynb`
- Make a copy of your notebook and associated files to a subfolder `original`. Make sure that the example code runs on your hardware and python configuration.

Network architecture

- **Task I:** Define your own custom neural network and train it. You may start from an existing architecture (e.g., copy PyTorch code of ResNet), but change or add at least one layer.

Dataloaders commonly return a tuple of input and label. For instance, for image classification, a pair of input image and its associated class label is returned. However, some tasks require various data sources and labels, and for others there is no explicit label (e.g., for autoencoders). To support any of these cases in your framework, we switch to named variables via dictionaries.

- **Task II:** Change the `__getitem__` function of your pytorch dataset class to return a dictionary. E.g., to `return {'img':img_tensor, 'class':class_tensor}`. You may want to write a wrapper to leave the original dataset class unchanged.
- **Task III:** Change the network to select the input from a dictionary passed on as an argument to the forward function, `def forward(self, input_dict)`, and to return a dictionary, e.g., return `{'class': c}`. This change will allow you to return and plot intermediate results needed for debugging without changing other parts of the code.
- **Task IV:** Extract the relevant tensors from the network output and data dictionaries to be passed on to the loss function. Alternatively, you can define a new loss that accepts dictionaries as input. Make sure that your network trains as before.

Evaluation

- **Task V:** Obey to the golden rule of ML. Split the data into training, validation, and test sets (one dataset and dataloader instance for each, unless already in that form). Write a function to evaluate the mean absolute error of your network. Apply it on the validation set, once after every epoch of training, and once on the test set after training.
- **Hint:** Use `torch.utils.data.Subset` to split an existing PyTorch dataset.
- **Hint:** You don't have to train till convergence in this assignment, 1000-5000 iterations are often sufficient to see correctness of implementation. Notably, running only on a few hundred iterations during debugging activity is advisable to speed-up development.

Visualization

- **Task VI:** Plot the input (image) and output of your neural network for one example batch in training and one from the validation set.
- **Task VII:** Plot the training loss as a function over gradient descent iterations.
- **Task VIII:** Plot the validation error, as a function of epochs.

Optional add-ons if you are curious (not graded):

- Share useful tutorials that you discovered on Piazza.
- Use the moving average to smooth the training error plot.
- Does the use of dictionaries slow down computation? Time it with and without!

- Make use of your GPU by moving computations with `.cuda()` to the GPU.

Pack your original and final notebook file and all dependent files (if you have) into a zip container. Name it `assignment1_firstName_lastName.zip`. Make sure that you marked your code with the `# Task N` labels specified above. If your code spreads over multiple files add a `# Task N in file example.py` pointer into the main assignment1.ipynb file such that we can find your changes easily. Moreover, the jupyter notebook that you submit must contain the cell output from a clean execution (restart kernel and run all cells sequentially), such that we don't have to execute the code for each submission.

Submit via Canvas (details will be announced on Piazza). Don't submit downloaded datasets and other temporary files (max 1 MB total submission size). Moreover, list at the top of your Jupyter notebook the tutorials that you used and report the source of your starting point (unless you started from the provided notebook).

Detailed list of resources

Editors:

- Jupyter Lab and Jupyter Notebook
  https://towardsdatascience.com/jupyter-lab-evolution-of-the-jupyter-notebo

PyTorch:

- PyTorch tutorial root: https://pytorch.org/tutorials/
- PyTorch introduction: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

Visualization:

- Tensorboard in PyTorch
  https://tutorials.pytorch.kr/intermediate/tensorboard_tutorial.html
- Tensorboard in colab
  https://medium.com/looka-engineering/how-to-use-tensorboard-with-pytorch-

Datasets (incomplete list):

- Custom dataset
  https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
- The famous MNIST dataset
  https://pytorch.org/docs/stable/torchvision/datasets.html#mnist
- Modern fashion MNIST variant
  https://pytorch.org/docs/stable/torchvision/datasets.html#fashion-mnist