Assignment 1: Playing with PyTorch

CPSC 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.

Setup python 3.7, PyTorch (version 1.6) 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

The problem. You start from the jupyter notebook `assignment1.ipynb` that implements digit classification. Its simplicity makes it easy to explore PyTorch's core functionality. Your main task is to extend it, such that the neural network takes two images as input and outputs whether the images are from the same class.

It will be helpful to study external pytorch tutorials and codebases. 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/...

Tasks I-III: Data handling. Datasets 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, our tasks requires multiple images and labels. To this and future cases in your framework, we switch to named variables via dictionaries.

- **Task I:** Change the `__getitem__` function of your pytorch dataset class to return a dictionary. E.g., `return {'img':img_tensor, 'class':class_tensor}`. You may want to write a wrapper to leave the original dataset class unchanged.
- **Hint:** Make yourself familiar with the PyTorch dataloader class, which iterates over the dataset to create batches. It should work on your dictionary variant without a change.
- **Task II:** 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 II:** 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.

Tasks IV-VI: Pairwise classification. First, make sure that the classification task still works after tasks I-III. We now turn the single digit classification into a pairwise classification.

- **Task IV:** Change the dataloader to return two images and a binary variable that indicates whether the images are the same. E.g., return `return {'img1':img_tensor, 'img2':img_tensor, 'matching_pair': matching_class_label}`
- **Task V:** Define your custom neural network to take in the two images and output a single output probability. You may start from the existing architecture working on single images, but at some point you have to merge/concatenate the information from both images.
- **Task VI:** Define a suitable loss function and train your neural network.

Evaluation.

- **Task VII:** Obey to the golden rule of ML. Split the data into training, validation, and test sets (one dataset and dataloader instance for each). Write a function to evaluate the accuracy of your network (compute the percentage of correct pairwise classification over the validation/test set). 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 VIII:** Plot the input images and output of your neural network for one example batch in training and one from the validation set.
- **Task IX:** Plot the training loss as a function over gradient descent iterations.
- **Task X:** Plot the validation accuracy, as a function of epochs. Add the final test accuracy as a horizontal line for reference.

Optional add-ons if you are curious or want to play save. Solutions to each add-on task give 0.5 bonus points that can offset points deduced in the main tasks. The maximum number of points is still capped at 5.

- Use the moving average to smooth the training error plot.
- Does the use of dictionaries slow down computation? Time it with and without!

Submission

Once finished, submit your jupyter notebook on Canvas, include everything in the same notebook. The jupyter notebook that you submit must contain the cell output (incl. plots!) from a clean execution (restart kernel and run all cells sequentially). If some of your outputs are displayed with external tools, such as Tensorboard, please include screenshots of those. Name your submission

`assignment1_firstName1_lastName1_firstName2_lastName2.ipynb` (or `.zip`). Do not submit downloaded datasets and other temporary files (max 5 MB total submission size). Moreover, list at the top of your Jupyter notebook the tutorials that you used.

Detailed list of resources

Editors:

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

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