Assignment 1: Playing with PyTorch

CPSC 533R Visual AI
by Helge Rhodin

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.

Setup python, PyTorch, 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 the conda package manager following the `readme.md`, or if you are on windows (not recommended) 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 shows how to track the training progress with Weights and Biases (W&B) link.

Tasks I-III: Data handling. Datasets commonly return a tuple of input and label. For instance, for image classification, a pair of input images and their associated class label is returned. However, our task requires multiple images and labels, and returning multiple tensors becomes confusing. Therefore, we switch to named variables via dictionaries.

- **Task I:** Create a dataset class that internally calls the provided MNIST dataset. Make the `__getitem__` function of this wrapper class return a dictionary. E.g., `return {'img':img_tensor, 'class':class_tensor}`.
- **Hint:** Make yourself familiar with the PyTorch dataloader class, which iterates over the dataset to create batches. It should work on your dictionary dataset variant without a change.
- **Task II:** Implement a neural network that operates on dictionaries. You can wrap pre-defines pytorch networks or build your own neural network. The forward function, `def forward(self, input_dict)`, must select the input image from the dictionary passed on as an argument and 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 III:** 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 `__getitem__` function to return a dictionary of two images and a binary variable that indicates whether the two images are the same.
- **Task V:** Define a suitable loss function to train your neural network on the task of identifying pairs of images that show the same number.
- **Task VI:** Define a custom neural network to take in the two images and outputs values compatible with your loss function from Task V.

Evaluation. Evaluate the accuracy of your model.

- **Task VII:** Obey to the golden rule of ML. MNIST comes with a train/test split but has no validation set. Split the training set into training and validation. Make sure that there is a clear split without overlapping images between the three. Write a function to evaluate the accuracy of your network as the percentage of correct pairwise classifications over the validation. Apply it on the validation set, once after every epoch of training, and once on the test set after training.
- **Hint:** You don't have to train till convergence in this assignment, 1000-5000 iterations are often sufficient to see the correctness of implementation. Notably, running only on a few hundred iterations during debugging activity is advisable to speed-up development.

Visualization. Make your evaluation visual.

- **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.

Bonus. Solutions to each bonus 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 *exponential moving average* to smooth the training error plot or integrate the W&B logging.
- Does the use of dictionaries slow down computation? Time it with and without!
- Optimize your model and train it for exactly 10 epochs. Report your accuracy number in percent on Piazza (we will create a post). You get 0.5 bonus points if within a 3% margin on the leader. The top-three will receive an additional 0.5 bonus points.

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). Mark the tasks in your solution with comments **#Task I**, **#Task II**, ... Moreover, list at the top of your Jupyter notebook the resources and tutorials that you used. Do not submit downloaded datasets and other temporary files.

Detailed list of resources

Editors:

- Jupyter Lab and Jupyter Notebook link
- PyCharm the most popular python editor (recommended for your course project)

PyTorch:

- PyTorch tutorial root: link
- PyTorch introduction: link

Visualization:

- Weights and Biases (W&B) link.
- Tensorboard (a decent alternative if you prefer not to upload your results to an external server): link

Datasets (incomplete list):

- Custom PyTorch dataset link
- The famous MNIST dataset link
- Modern fashion MNIST variant link