SER Practicals - Day 2

Today we're going to implement the training part of the code! You will be given most of the working code (you've had plenty of deep learning modules before this afterall) and your job will be to refactor it keeping in mind all of the principles we've covered so far. You should be able to call the training algorithm from the cli tool and configure all hyperparameters via the cli. We will review your code via pull requests on GitHub.

Getting the latest version of the code

If you were working on the main branch you will now find that your copy of main does not track the remote. Uh oh! Lesson learned, always work on a branch ;).

You can fix this pretty easily though. If you want to keep your day one solutions, simply run:

```
git checkout -b day-1-solutions
git commit -m "my solutions for day 1"
```

You'll then have a branch with your day 1 solutions to look back on fondly.

Then, checkout out the main branch again and run:

```
git fetch --all
git reset --hard origin/main
```

We didn't cover those two commands but it will basically set your local main to match the remote main exactly. In reality you will almost never want to do stuff like this but in this case it makes sense.

Be especially careful with git reset, it's one of the very few ways you can actually lose work with git. If you have uncommitted changes on your local main branch they will be lost after running that command.

Implement Training the SER way

1. Pull the latest code from the upstream remote
    - You will need to install the extra dependencies - `pip install -e .`
    - You will find in cli.py all the code necessary for training an mnist model
    - It runs and trains well!
    - But it's ugly, let's change that…
2. Make all hyperparameters inputs via the cli using Typer
    - Hint: there's an example in the code
3. Create a new directory called ser in the project root.
4. Inside that directory, create a model.py, a train.py, a data.py, and a transforms.py file
5. Refactor the train() from cli.py into their separate components
    - Remember to commit often!
6. Make sure the code still works
    - Tedious isn't it? We'll see how we can make this less tedious on another day.
7. Can you spot the horrendous inefficiency?
    - It's not an error per se but it's not necessary and it slows things down
    - Hard to spot when the code is all in one block!
8. Once you are happy, get your code into GitHub and make a pull request for review
9. Make any necessary review changes until the code gets accepted

Experiment Tracking

Our model trains but we have nothing to show for it! We need to save our trained models and the hyperparameters we used to define them.

1. Decide on a directory structure for tracking each experiment and each run of an experiment
    - Hint: what properties uniquely identify a particular run of an experiment? The name? The hyperparameters? The date? The day of the week?
2. Save the model once you're done training in the run directory
    - PyTorch has a handy oneliner for this
3. Save all the hyperparameters that were passed in via the cli to a json file in the run directory
4. Submit code for a review via pull request
5. Make any necessary review changes until the code gets accepted

Challenge

1. Record the highest validation accuracy for each run
    - It's a good idea to save these in the same place as all the other run information
    - You could start by recording the last validation accuracy
    - Then try to record the best validation accuracy and matching epoch
    - Perhaps record the accuracies for all epochs but only save the model which had the best validation accuracy
2. Use a dataclass to pass around the parameters
    - We briefly touched on dataclasses in the lecture. See https://realpython.com/python-data-classes/ for more info.
3. Save the git commit hash in the hyperparameters file
    - Hint: there's a library called gitpython which will make this really easy.
4. Try another, more standard model
    - Torchvision has standard models that you can load without defining the architecture. Try one standard model
    - This isn't about making ML code better, but about seeing how we can switch models seamlessly in our well abstracted code