NEURO 140/240

Tutorial 1 - Intro

Agenda

- Tools of the trade (Conda, Jupyter, Git)
- Quick introduction to machine learning + PyTorch
- Further Questions + DIY

Agenda

- Tools of the trade (Conda, Jupyter, Git)
- Quick introduction to machine learning + PyTorch
- Further Questions

Tools of the Trade: Conda

- Most popular data science / scientific computing package manager

- Helpful for:
  - Managing multiple versions of Python
  - Quickly installing/uninstalling packages/Python
  - Separating your NEURO 140 project from other Python projects

Tools of the Trade: Conda

Install from https://docs.conda.io/projects/conda/en/stable/user-guide/install/index.html

- (You probably want Miniconda or the Anaconda Distribution, either is fine)

Once installed, open a terminal. You should see a (base) next to your terminal line

```
(base) ubuntu@valerio:~$
```

To create a new environment, type ‘`conda create -n NAME python=3.x`’
To use that environment, type ‘`conda activate NAME`’

To install new packages, type ‘`conda install PACKAGE_NAME`’

- If this really doesn’t work, try using ‘`pip install PACKAGE_NAME`’ instead: it’s not best practice (pip is a separate package manager) but this usually works

Tools of the Trade: Conda

Common Conda Issues:

1. Don't start a Python environment with the newest version of Python (3.13 as of these slides): many packages will <u>not</u> be available, including some fairly common ones, so you'll run into issues
   a. For this class, I recommend Python 3.11 or 3.12
2. Some packages are fussier about dependencies than others: the fussiest is usually PyTorch, so it's good practice to install that first, right after creating the empty environment
3. If you already have Conda, make sure it's up-to-date (the current version is 25.3)

Tools of the Trade: Jupyter

Jupyter Notebooks (or general Python notebooks, .ipynb) allow you to split your code up into “cells”, each of which you can run separately from the others

Really good for iterating quickly over software, since e.g. you don’t have to load up and process all of your data again every time you re-run your ML model

```python
print("hello world im a cell")
```

Python

```python
print("hello world im a cell too")
```

Python

i'm a cell too, and I'm not even code!

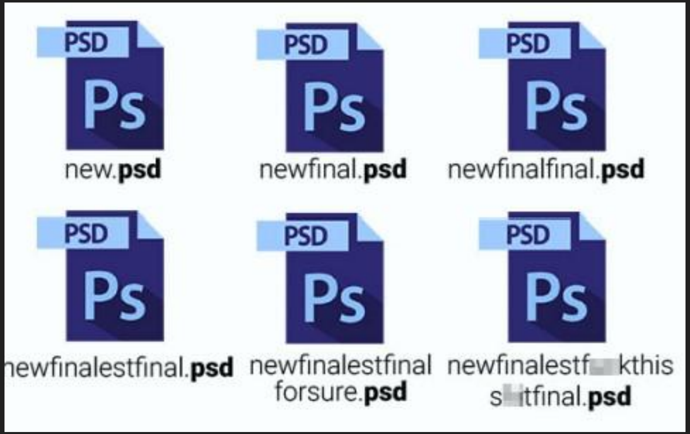
Tools of the Trade: Jupyter

Tips/Potential annoyances:

- Conda + Jupyter together can sometimes be problematic; always install the ipykernel package into your Conda environment so you can run cells in notebooks
- **Careful when deleting cells**: they can be very difficult to get back (as opposed to a regular Python file), and if you don't split up your code enough you can easily delete most of your work with a single keystroke
- Jupyter lends itself really well to graphing through packages like matplotlib, since you can quickly change elements of a graph and re-run a cell – be sure to use this in your reports!

Tools of the Trade: Git

Git is a version control system, to avoid the following:



Useful for having concrete, working versions of your code you can go back to if anything goes south. Probably not going to be incredibly useful for this project unless you are doing something large-scale (but you may be!)

Tools of the Trade: Git

GitHub is a popular implementation of Git we recommend using.

Useful commands:

<u>Group 1: Getting stuff from GitHub</u>

git clone - Copy from GitHub to local.

git fetch - get changes from remote

git merge - merge changes into local

git pull = git fetch + git merge

<u>Group 2: Reflecting your local changes on GitHub</u>

git add - add a changed file to staging area

git commit - commit the change i.e. create snapshot

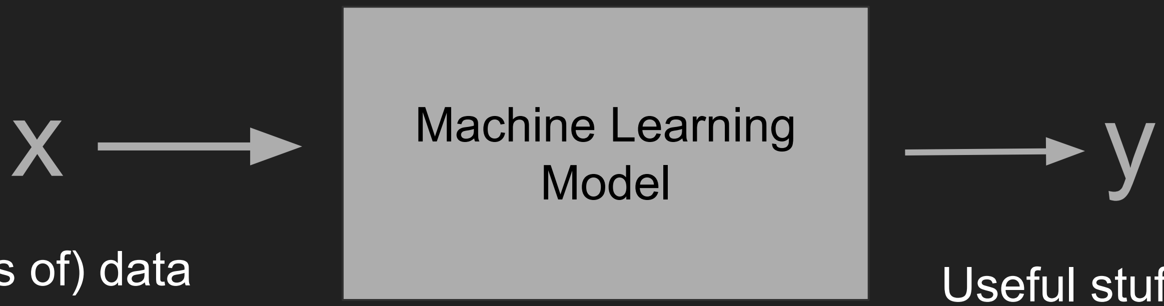
git push - upload new snapshot to github

Agenda

- Tools of the trade (Conda, Jupyter, Git)
- Quick introduction to machine learning + PyTorch
- Further Questions + DIY

Intro to ML

The Machine Learning Paradigm



(Lots of) data
comes in

The data helps train
some mapping from
input to desired output

Useful stuff comes
out (classifications,
regressions)

Intro to ML

How do machines learn?

TL;DR: ML Models are just a bunch of matrix multiplications with non-linearities in between. These matrices are randomly initialized, and during training, we change the values of each matrix depending on how much that value contributed to mistakes in the model.

(♪ blackboard interlude ♪)

(regression example + why do we need non-linearities)

Intro to ML

What can machines learn?

Supervised Learning

Types of tasks where you have access to input-output pairs (x,y)

(e.g. regression, some classification)

Unsupervised Learning

Types of tasks where you have access to input data and you want to extract patterns from it

(e.g. clustering, autoencoders)

Reinforcement Learning

Types of tasks where you can define some “agent” working in some “environment”, where actions they can take have different rewards

(e.g. playing chess)

Intro to ML

How can I make machines learn?

Very powerful, very easy to use (as compared to other ML libraries at the time), extensible and usually very good documentation

Quick Intro to PyTorch:
https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html

Intro to ML

How can I make machines learn?

Getting the data is the hardest part: defining and training the model itself is very few lines of boilerplate code:

```
model = nn.Sequential(
    nn.Linear(28*28, 64),
    nn.ReLU(),
    nn.Linear(64, 64),
    nn.ReLU(),
    nn.Linear(64, 10)
)
```

```
epoch_num = 5
for epoch in range(epoch_num):
    losses = []
    accuracies = []
    for batch in train_loader:
        x, y = batch

        b = x.size(0)
        x = x.view(b, -1)


        l = model(x)
        J = loss(l, y)
        optimiser.zero_grad()
        J.backward()
        optimiser.step()

        losses.append(J.item())
        accuracies.append(y.eq(l.detach().argmax(dim=1)).float().mean())
    print(f"Epoch {epoch+1}/{epoch_num} - Loss: {J.item():.4f}, Accuracy: {accuracies[-1].item():.4f}")
```

Intro to ML

How do I choose an architecture/loss function?

In many cases, it depends entirely on the task; no choice needed, just think about a suitable metric of ‘error’.

Regression? Use mean squared error
Classification? Use cross-entropy loss (a measure of how confident the model is about a prediction)

Something else entirely? PyTorch allows you to make your own!

(Or there are dozens more already defined: Margin Loss, KLDiv Loss, etc.)

Intro to ML

How do I choose an architecture/loss function?

Same goes for the architecture:

FCNN/“Linear”: Anything, but usually for regressions, or classification of things that are not images.

CNN: Object recognition (that’s what it’s optimized to do!).

Transformer: Sequence -> Sequence problems, e.g. Language, Time Series

RNN/LSTM/GRU: Also Sequence -> Sequence; older than transformers, probably more powerful but slower/harder to train

Intro to ML

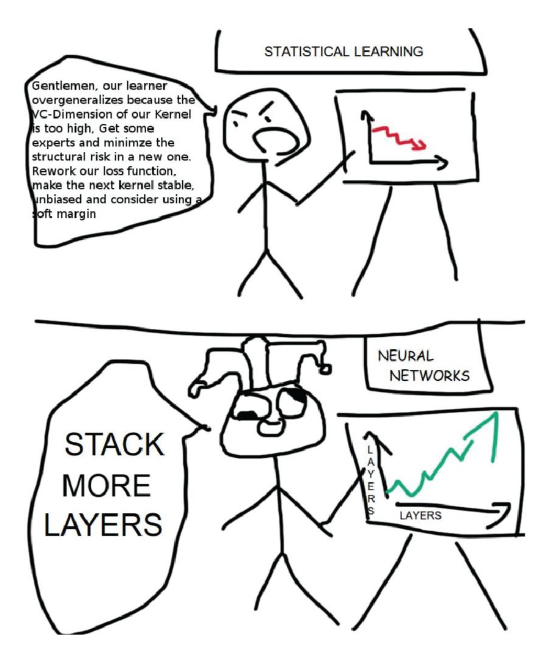
How do I choose activation functions / number of layers / hyperparameters?

Experiment. In most cases, ReLU is good as an activation function.

Number of layers depends on how complex your data is, and has to be empirically determined.

Hyperparameters are similar: move values around until they work, be careful about setting the learning rate too high.

SGD or Adam are the canonical optimizers, you shouldn't need to use anything else.



Intro to ML

(♪ MNIST Example Interlude ♪)

Intro to ML

General things to be aware of:

1. If you have a GPU, use it – CPU training can be quite slow, and PyTorch is optimized for GPUs (consider using Google Colab if you don't)
2. Careful about choreographing the J.backward() and optimizer.zero_step(): if you put them in the wrong order (which depends on your training loop) your model won't learn. If you have problems with your model learning, try swapping them as a sanity check: this has worked wonders for me in the past
3. Never train on the val set, be careful to always use something like torch.eval() to make sure you're not updating gradients on it – this can trick you into thinking your model's better than it is!

Agenda

- Tools of the trade (Conda, Jupyter, Git)
- Quick introduction to machine learning + PyTorch
- Further Questions + DIY

Feel free to contact me at valeriopepe@college.harvard.edu w/ any questions (or Ed!)