Debugging ML Code

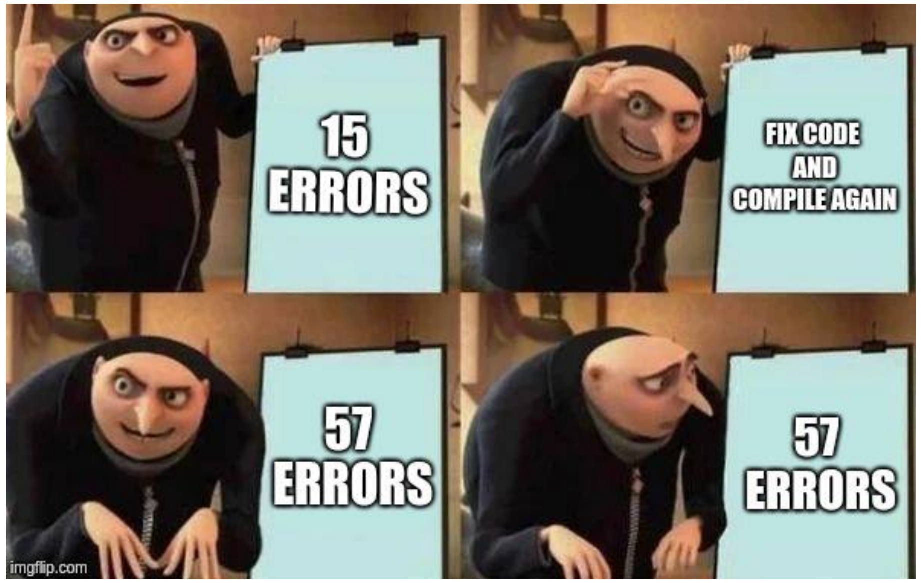
02457 Machine Learning Operations

Nicki Skafte Detlefsen,

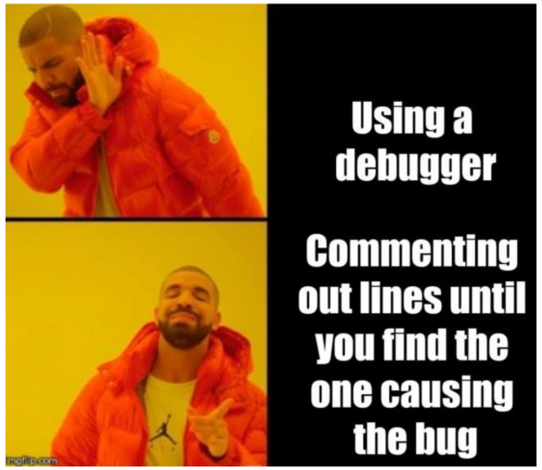
Postdoc

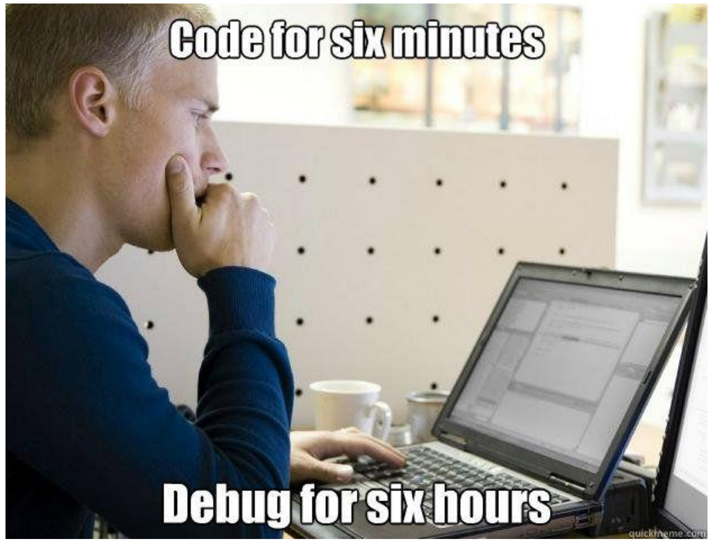
DTU Compute

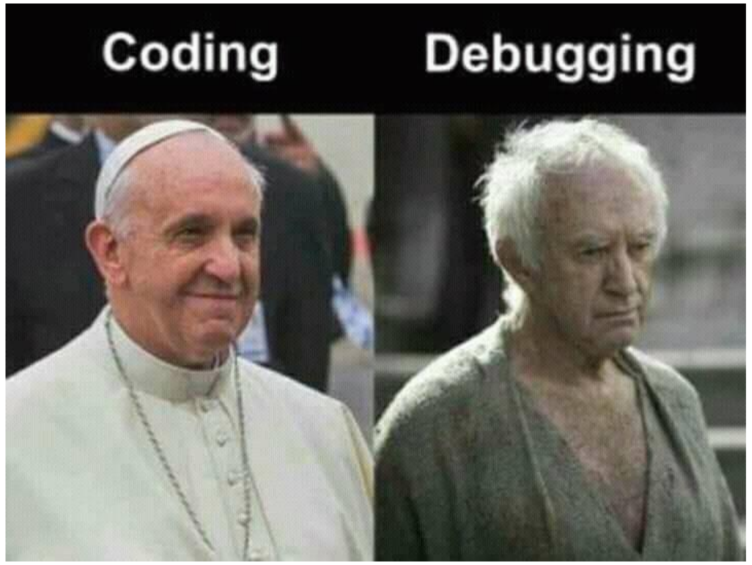
Dubugging is hard but nessesary

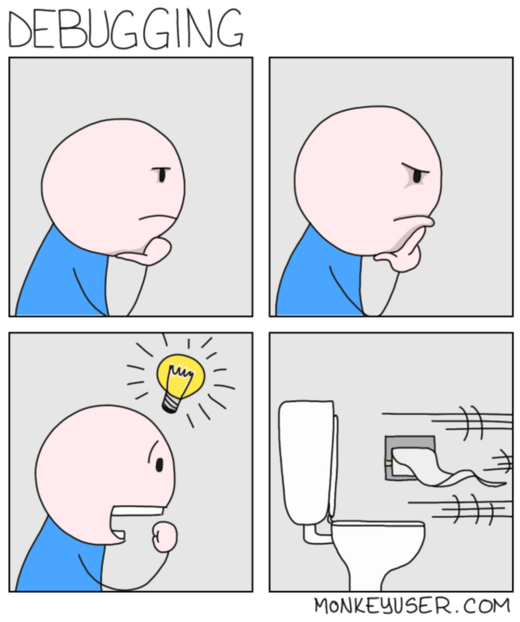


DEBUGGING
I DON'T KNOW WHERE YOU ARE, I DON'T KNOW HOW YOU WORK, BUT I WILL FIND YOU, AND
I WILL FIX YOU









First step: Get a good enviroment

Notebooks considered mostly harmful

I know that iPython notebooks are popular and for good reason: It is easy to get started, and you don't have to worry about too much "system stuff". But eventually, the issues with notebooks outweigh the benefits. So for you own sake: get a "real" editor and work on proper scripts.

Spyder

PyCharm

Visual Studio Code

Torchstudio
(in beta)

3 ways of debugging python code

1. Print statements in your code (yes, seriously):

```
print('x.shape = {}'.format(x.shape))
```

2. Stop at an interesting point in your code and interact with the code

```
from IPython import embed; embed()
... # do your stuff interactively here
exit() # exit ipython to let your code continue
```

3. Work in an actual debugger

```
import ipdb; ipdb.set_trace()
```

See details here: https://switowski.com/blog/ipython-debugging

When everything is running, but results are wrong

MLOps

Debug the math!

Go through your code, and if you cannot do a one-to-one match between equations and particular lines of code, then rewrite the code (that will create good habits for you to follow).

Check dimensions!

The number one bug is that people get tensor dimensions wrong.

Solution: Annotate your code with comments about the shapes of all variables.

```
A = torch.randn(N, D)  # NxD
x = torch.randn(D)  # D
Ax = A.mv(x) # N
```

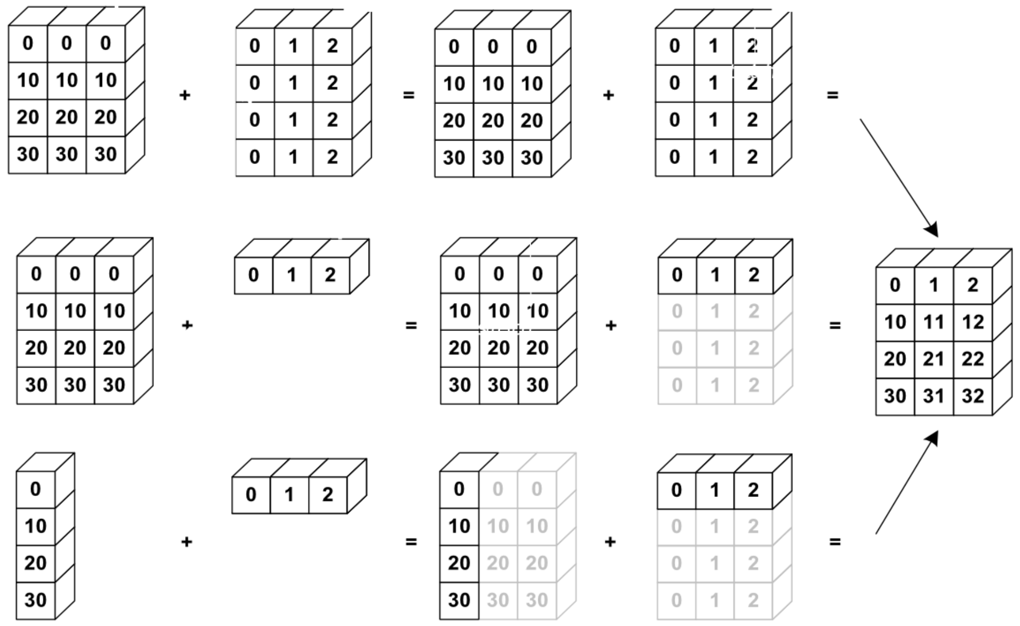
When everything is running, but results are wrong

Lookout for broadcasting!

Broadcasting is both a blessing and curse of python

It can create serious problems (real life example)

```
preds = torch.randn(100)
target = torch.randn(100,1)
loss = (target – preds).abs().pow(2.0).sum()
```



What is the problem here?

Software can help

You can go further and use ShapeGuard:

```
pip install torch-shapeguard
```

Then you can do

```
A.sg(N, D)
x.sg(D)
Ax.sg(N)
```

and your code will fail if dimensions aren't right. Quite neat!

See details here: https://github.com/rasmusbergpalm/shapeguard

Reduce Model Complexity

When things don't work, build a more simple model (ideally start with a simple model)

If you can, start with linear model. Here you often understand all aspects of the model. For instance, an autoencoder with a linear encoder and decoder should give you the same latent codes as PCA. Always, start with the settings where you can verify if your implementation does the right thing in non-trivial models, we don't know what the model should be doing, so debugging is difficult...

Plotting

1. Can you reduce data dimension (synthetically) ? That might allow you to plot stuff.

2. Can you reduce the number of model parameters? That might allow you to plot how the parameters evolve during training.

3. If you cannot reduce dimensions (are you sure?), then plot anyway. Beware of t-SNE and the likes: plots looks great but tend to be so misleading that you cannot learn from them.

Why do I get Nans?

An all too common issue we run into is that our code produces NaN. Generally speaking code produces NaNs if you multiply/divide by annoying combinations of Inf and 0.

Step 1: locate where that happens.

Often you cannot avoid these situations, and you should consider working on a log-scale.

Step 2:

```
a = 1
for x in data:
   a = a * x
```

```
log_a = 0
for x in data:
   log_a = log_a + log(x)
```

Trade-off between precision and stability

Summary

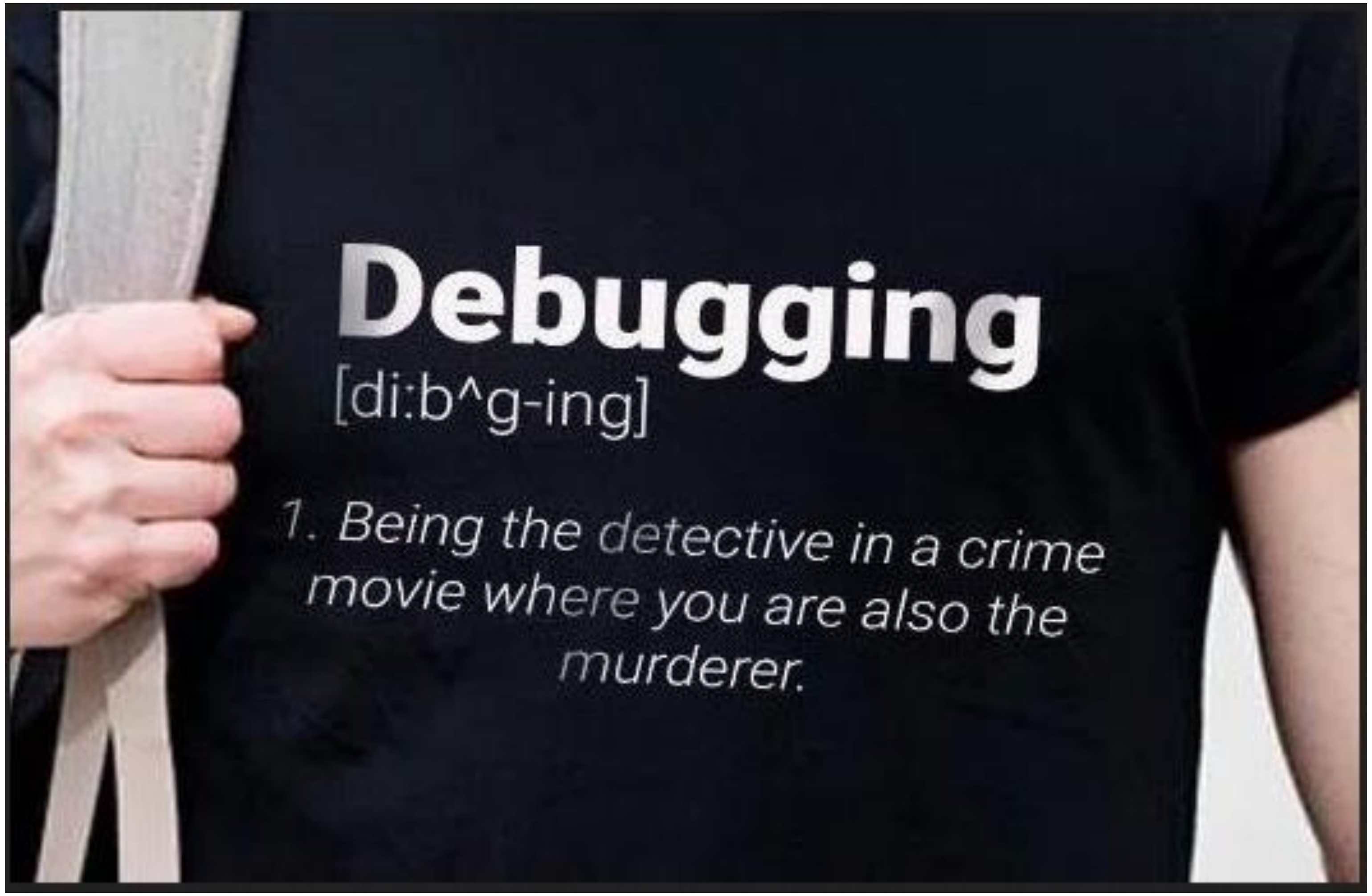
1. Debug the math (check dimensions, redo derivations, ...)

2. Reduce model complexity (aim for a linear model)

3. Plan your next plot (keep plotting trustworthy)

4. Avoid NaNs on a log-scale

Meme of the day


Debugging
[di:b^g-ing]
1. Being the detective in a crime movie where you are also the murderer.