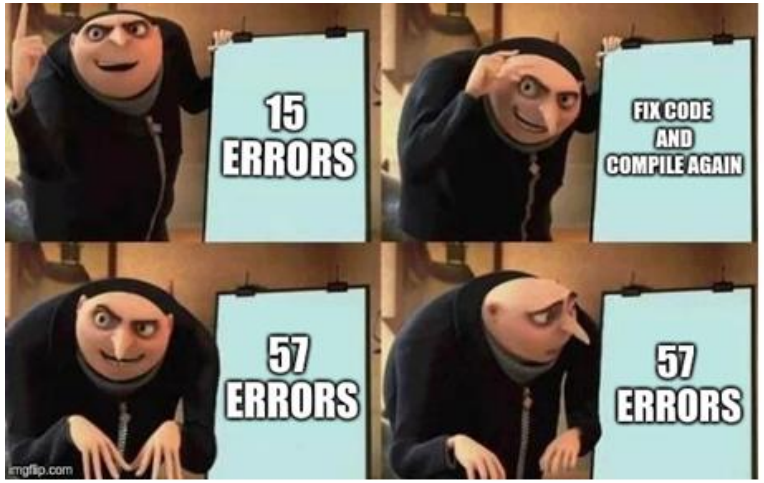
02476 Machine Learning Operations

Nicki Skafte Detlefsen

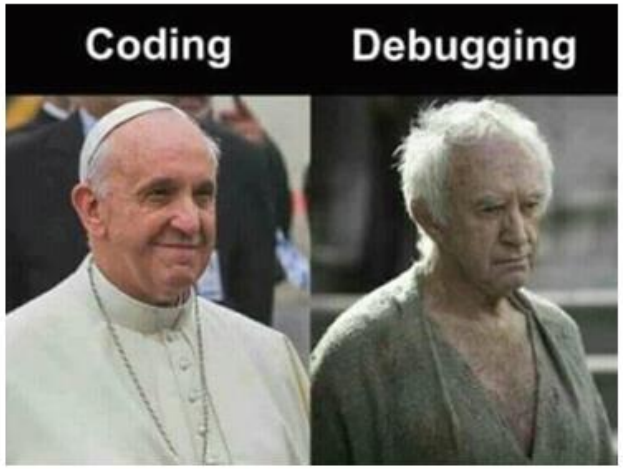
# Debugging ML code

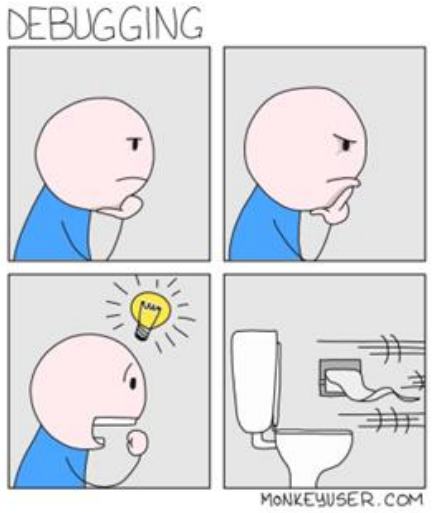
# Debugging is a hard but necessary disiplin









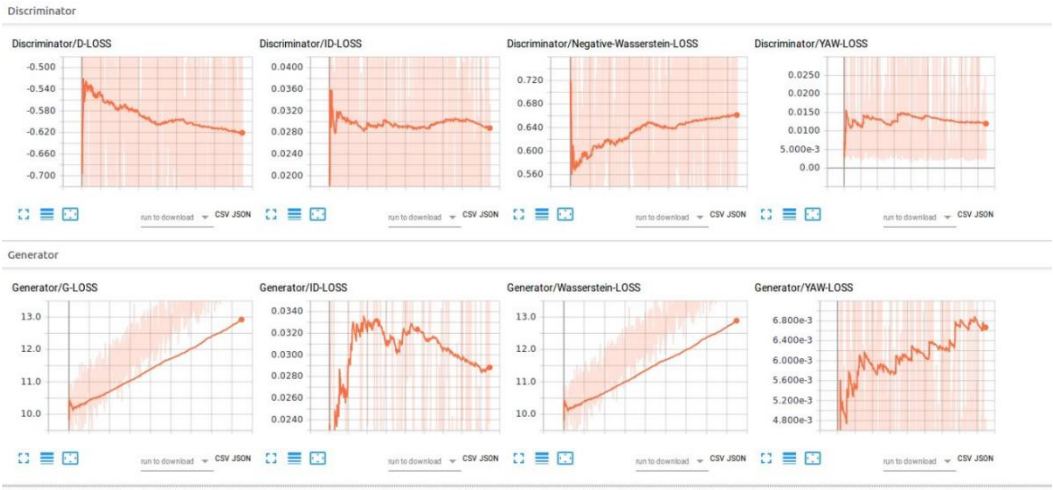
# Debugging ML code is even harder

Bugs in ML code can be non-ml specific and ML specific

- Classic bugs: Code does not run
  - Use traditional debugger to find these

- ML specific bugs: My model is not converging
  - Need the correct approach for debugging

Lets start with the classics…



# First step: Get a good environment

💡 Jupyter notebooks are great at what they are meant for: exploring ideas and combining code + text into standalone document…

💡 However, it can be a pain debugging code in notebooks…

VS Code

Torchstudio


Pycharm


Spyder

# Python debugging in general

1. Print statements

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

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

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

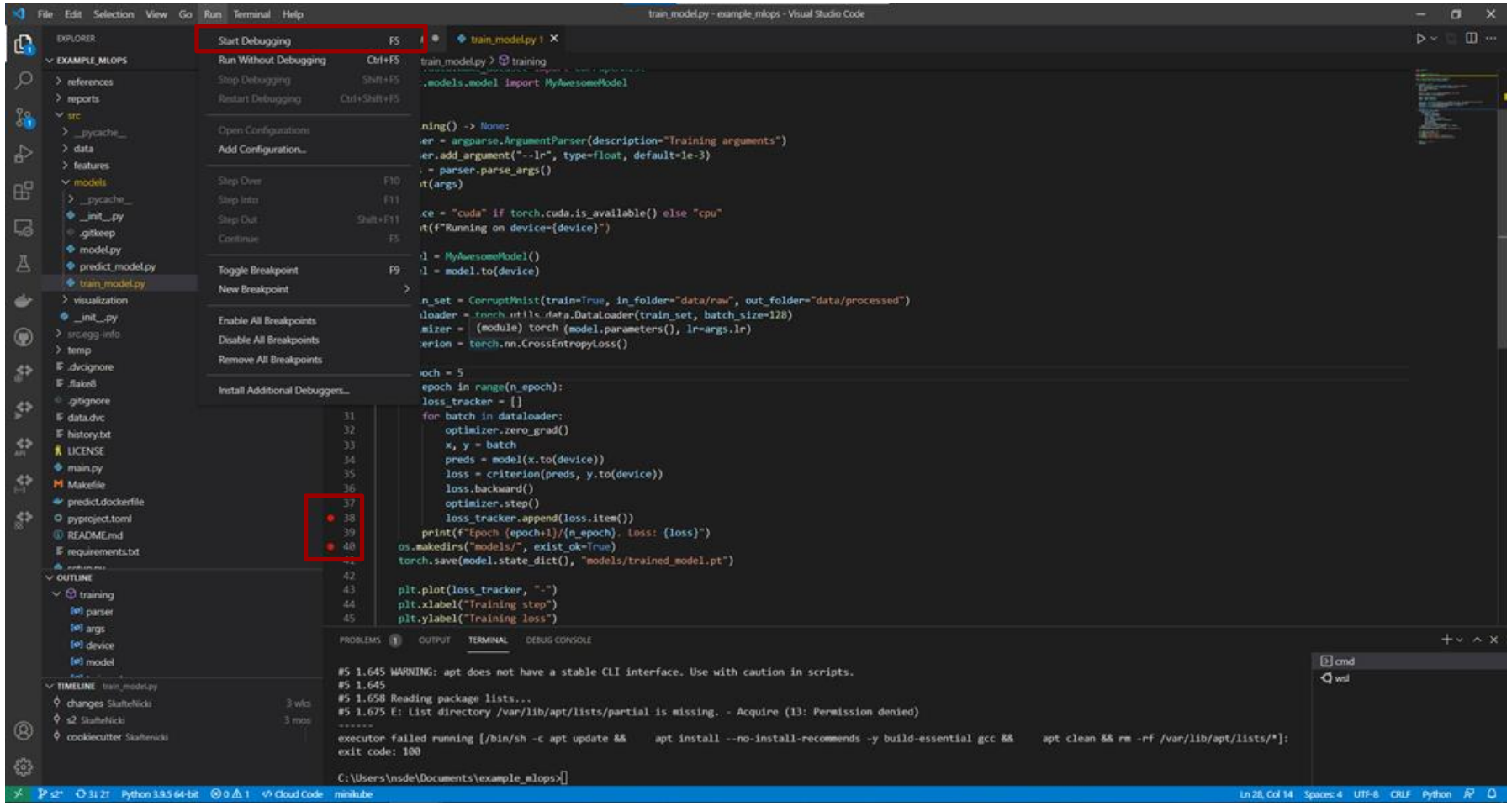
3. Work in an actual debugger

```
import pdb; pdb.set_trace()
```

Learn more here

https://switowski.com/blog/ipython-debugging

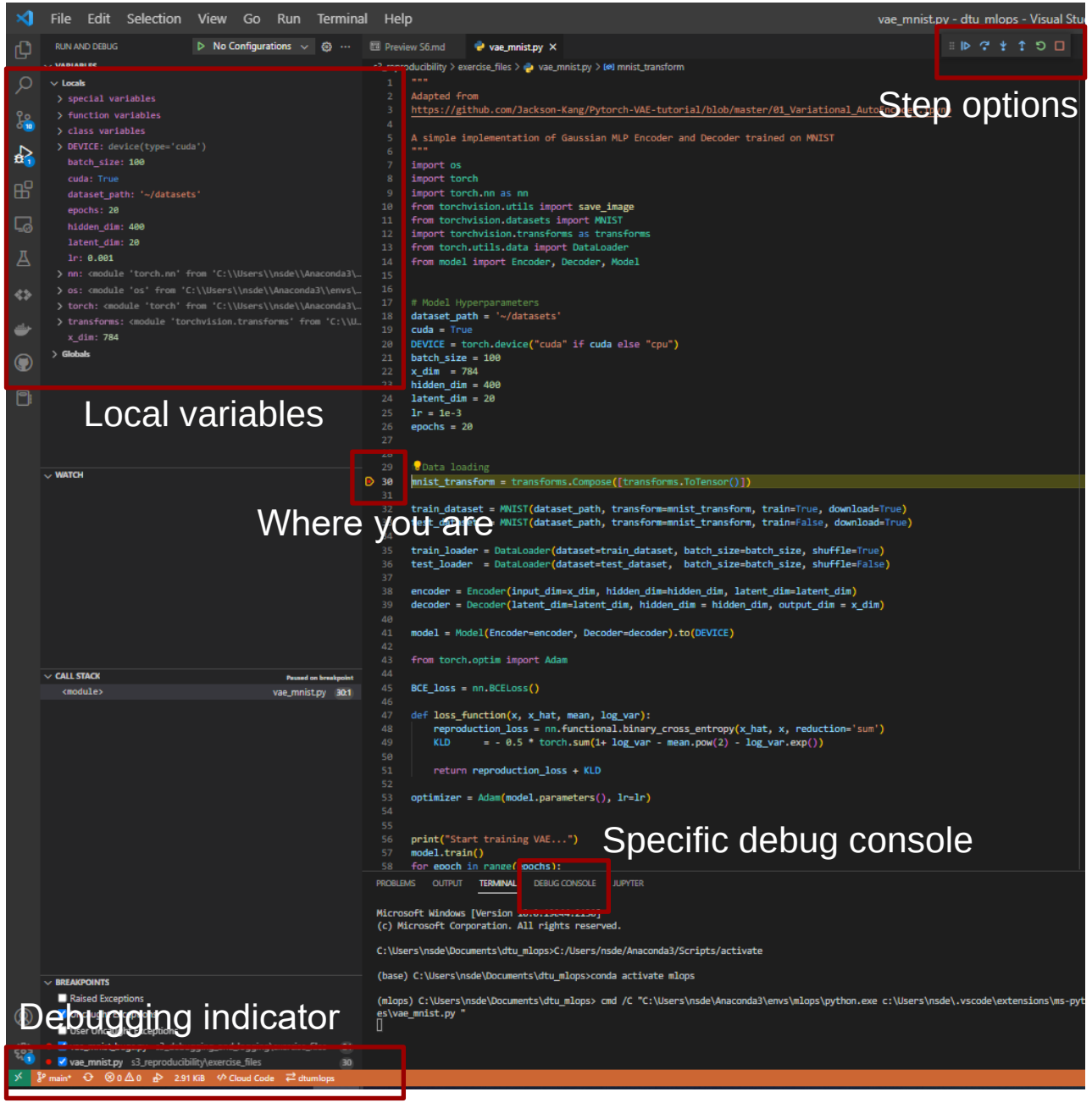
# VS code debugger



# VS code debugger

Step options:

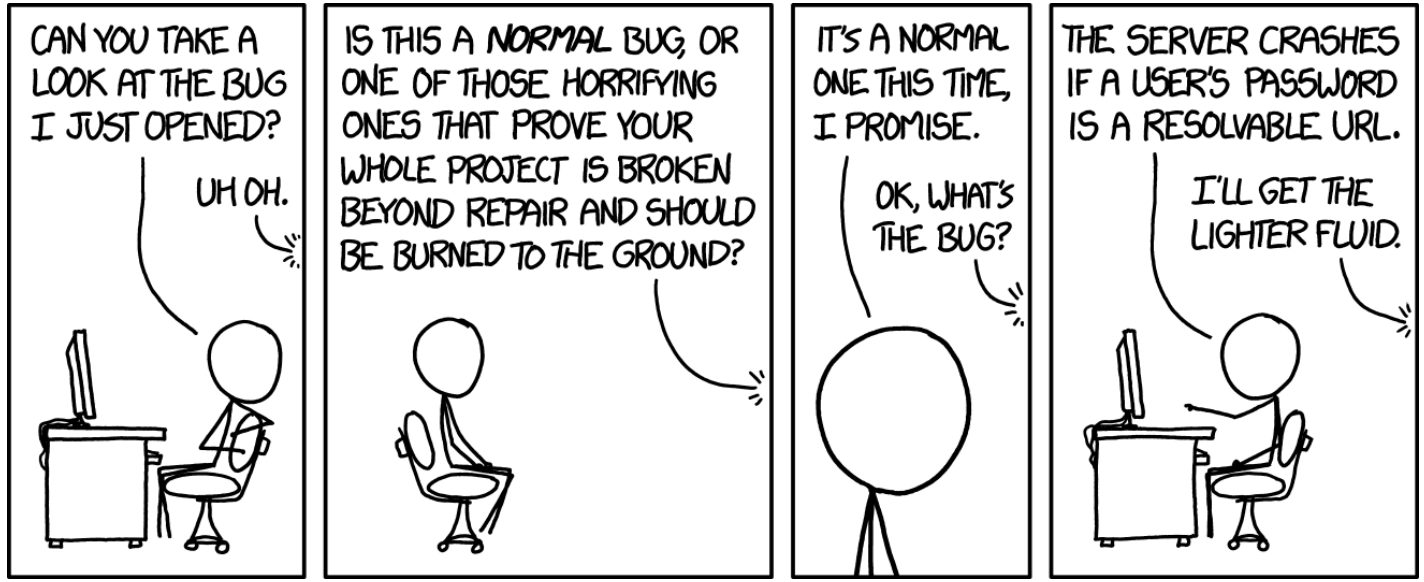
- F5: next breakpoint
- F10: next line
- F11: step into
- Shift-F11: step out
- CTRL-Shift-F11: Restart
- Shift-F5: stop



# Back to these ML specific bugs

Or better know as

⚠ When everything is running, but results are wrong ⚠




Finding these buggers comes with experience

A potpourri of my findings over the last couple of years and others.

# 1. Check your data!

Your starting point should always be the data in ML!

Check

- ☑ Examine summary statistics (data normalized?)
- ☑ Look at label distributions (is it shuffled?)
- ☑ Visualize a few samples (are they corrupted in anyway?)

If you are working on datasets you know, you may skip these steps.

# 2. Start as simple as you can

Remove all the fancy stuff

- Mixed precision
- Regularization like dropout
- Early stopping
- Learning rate schedulers
- …

One or more of these may be enabled by default if you are starting from someone's else codebase

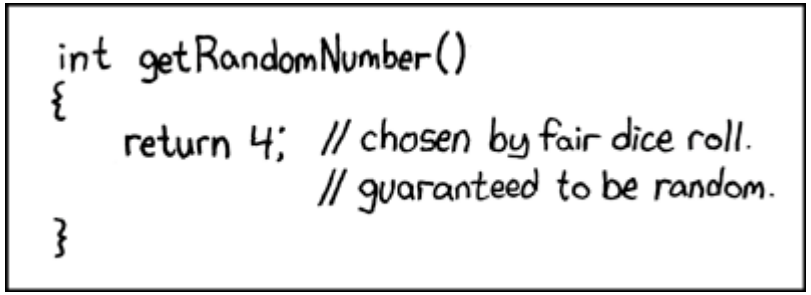

Prompt:

*Machine Learning bot with a fancy hat*

# 3. Make everything deterministic

Every machine learning run is by default random. Try fixing:

- Seed everything and use same seed everywhere
- Remove all data augmentation
- Use only a single batch of data where you have a feeling of the outcome



# 4. Investigate the math

Debug the math:

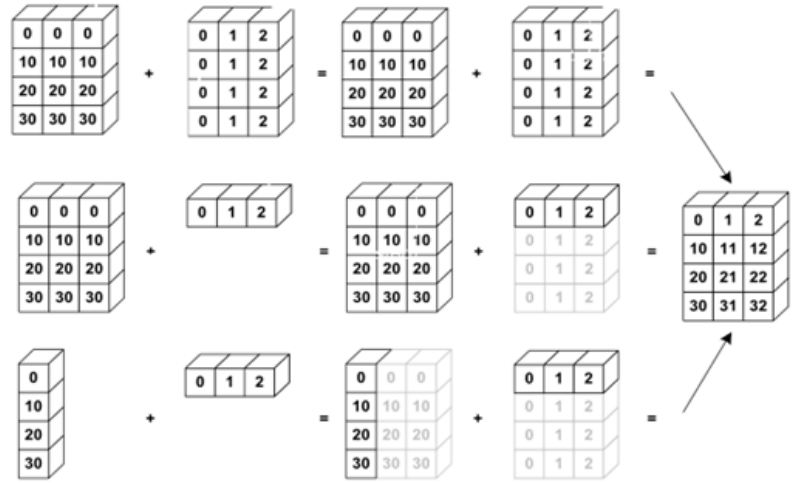
- 💡 Go through your code, line by line
- 💡 There should be a one-to-one match between equations and lines of code
- 💡 Refactor if it is not clear

Check dimensions and annotate if necessary or use typing software

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

# 4. Investigate the math (continue)

- Lookout for broadcasting!
- Broadcasting in python is both a blessing and a curse
- It can create problems (real life example)



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

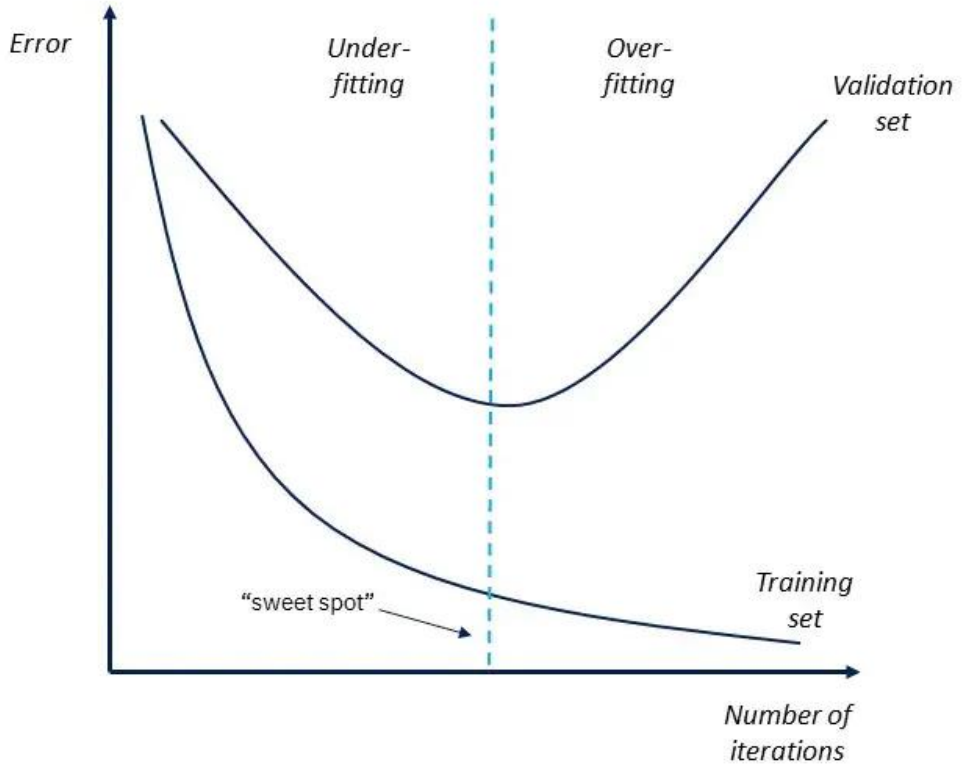
What is the problem here?

# 5. Overfitting is good?

Overfitting is usually seen as an bag thing bug…

💡 Models should be able to memorize one batch

💡 Train on one batch, if loss is 0 move on to larger models/large data else debug



# 6. Really look at your loss

Assuming your model is training without errors, but your loss is not behaving as it should.

- Are you printing/logging the results correctly?
- Did you remember loss.backward, optimizer.step and optimizer.zero_grad ?
- What about your learning rate and batch size?

For your loss, if possible, calculate in log-space

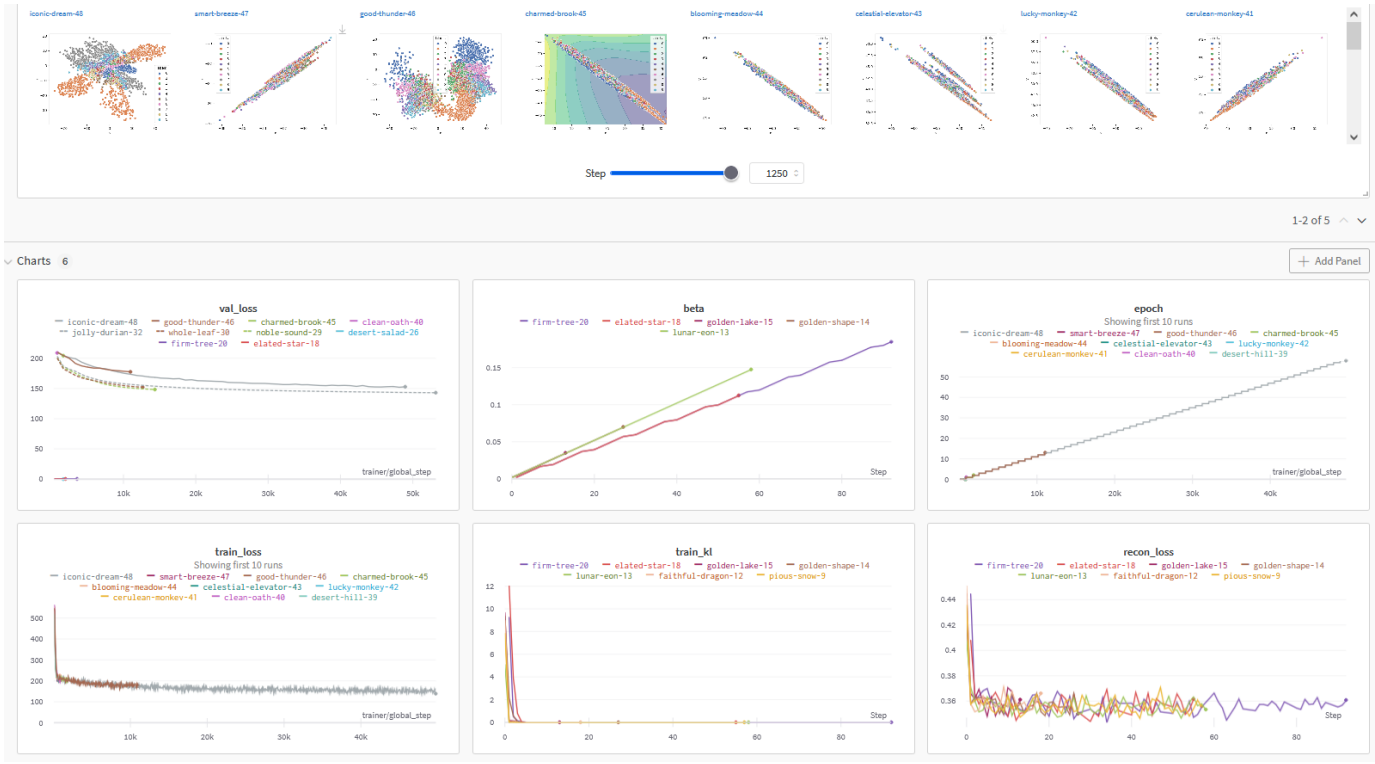
```
a = 1
for x in data:
  a = a * x

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

# 7. Visualizations are your friend

Log and visualize everything

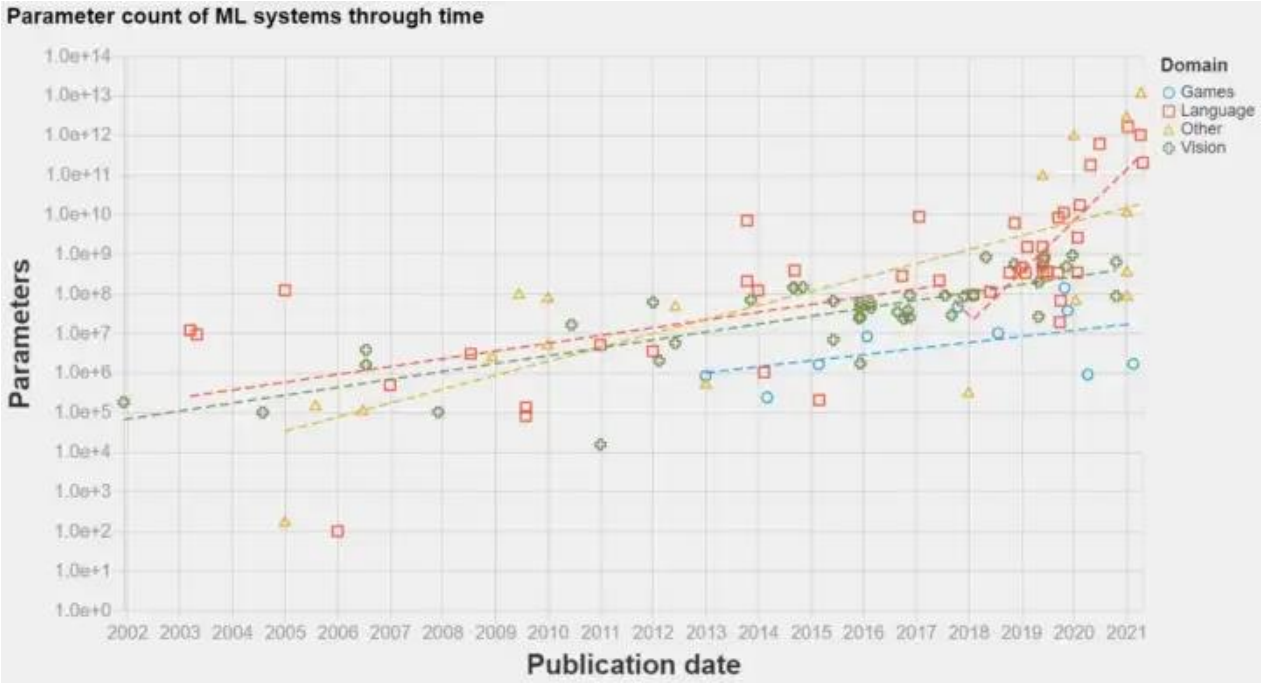
- 💡 Training loss is really decreasing right?
- 💡 Can you add additional metrics?
- 💡 Log dynamic changing hyperparameters (learning rate with lr schedulers)
- 💡 Plot data, predictions, reconstructions etc. over time



# 8. Add complexity over time

If you are still good, then

- Get a baseline that just works (=train on more data data)
- Stop overfitting:
  - Add back regularization
  - Add back data augmentations
- Tune hyper parameters



# Summary of steps in ML debugging

1. Check your data
2. Start as simple as you can
3. Make everything deterministic
4. Investigate the math
5. Overfit to your data
6. Look at your loss
7. Visualize everything
8. Add complexity in steps

# Meme of the day