

02476 Machine Learning Operations Nicki Skafte Detlefsen

Debugging ML code

Debugging is a hard but necessary disiplin



MONKEYUSER.COM

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...

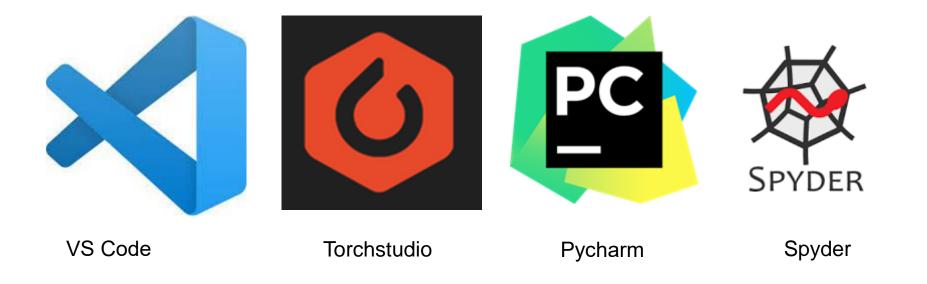




First step: Get a good environment

 $\[Pi]$ Jupyter notebooks are great at what they are meant for: exploring ideas and combining code + text into standalone document...

 $\ensuremath{\mathbb{Q}}$ However, it can be a pain debugging code in notebooks...



Python debugging in general

1. Print statements

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

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

3. Work in an actual debugger

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

import pdb; pdb.set_trace()

Learn more here

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



VS code debugger

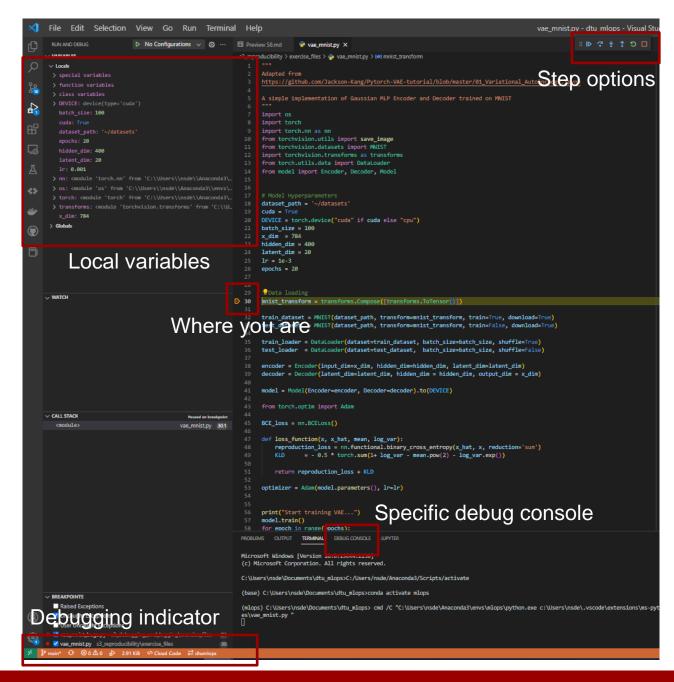
10	🖞 File Edit Selection View Go Run Terminal Help train_model.py - example_mlops - Visual Studio Code					- a ×
d'	EXPLORER	Start Debugging		I 🔹 🔹 train_model.py I 🗙		▷ ~ □ □ …
-	V EXAMPLE_MLOPS	Run Without Debugging	Ctrl+F5	train_modeLpy 2 ⊕ training		
0	> references			wordels model import MyAwesowelodel		The second second
1.1	> reports					All and a second s
8						18E
1.000				<pre>ning() -> None: .er = argparse.ArgumentParser(description="Training arguments")</pre>		and a
4	> data	Add Configuration		er.add_argument("lr", type=float, default=le-3)		
	> features			i = parser.parse_args()		
ß	✓ models			it(args)		
	_pycache			ce = "cuda" if torch.cuda.is available() else "cpu"		
50	 gitkeep 			<pre>rt(f"Running on device=[device]")</pre>		
0.000	modeLpy					1
因	predict_modeLpy	Toggle Breakpoint	F9	1 = MyAwesomeWodel() 11 = model.to(device)		
		New Breakpoint		of = molentia(gevice)		
÷.	> visualization	reew oreactions		<pre>_n_set = CorruptMnist(train=True, in_folder="data/raw", out_folder="data/processed")</pre>		
	🤹initpy	Enable All Breakpoints		<pre>iloader = trach intls_data_DataLoader(train_set, batch_size=128)</pre>		
		Disable All Breakpoints		<pre>mizer = (module) torch (model.parameters(), lr=args.lr) arion = torch.nn.CrossEntropyLoss()</pre>		
	> temp	Remove All Breakpoints		er nur – turcul militar (1955) (1997) (1997)		
_5>	E.dvcignore			och = 5		
	F .flake8	Install Additional Debugge	*S	epoch in range(n_epoch):		
_\$>	 gitignore ₣ data.dvc 			loss_tracker = [] for batch in dataloader:		
1.17.20	F history.bd			optimizer.zero_grad()		
<u></u>	I UCENSE			x, y = batch		
	main.py			<pre>preds = model(x.to(device)) loss = criterion(preds, y.to(device))</pre>		
	M Makefile			loss - craterion(press, y-to(uevace))		
100	predict.dockerfile			optimizer.step()		
5°	o pyproject.toml			loss_tracker.append(loss.item())		
1997 - 1997 -	README.md			<pre>print(f"Epoch {epoch+1}/{n_epoch}. Loss: {loss}") .makedirs("models/", exist_ok=True)</pre>		
	F requirements.bd			<pre>cch.save(model.state_dict(), "models/trained_model.pt")</pre>		
1	V OUTLINE					
	✓ ♀ training			t.plot(loss_tracker, "-") t.xlabel("Training step")		
	(P) parser			<pre>cutated(realing step) trylabe('realing los")</pre>		
	(e) args			OUTIVIT TERMINAL DEBUG CONSOLE		+ • ^ ×
	(e) device					
_	el model			NING: apt does not have a stable CLI interface. Use with caution in scripts.	D cmd	
	V TIMELINE train_model.py		#5 1.645			
	changes SkutteNicki			ding package lists List directory /var/lib/apt/lists/partial is missing Acquire (13: Permission denied)		
8	♦ s2 SkatheNicki					
	cookiecutter Staffenicki		executor fail exit code: 10	led running [/bin/sh -c apt update && apt installno-install-recommends -y build-essential gcc && apt clean && rm -rf 30	/var/lib/apt/lists/*]:	
£63						
				e\Documents\example_mlops>]		
*	P s2* O 31 21 Python 3.9.5 64-bit	© 0 ∆ 1 ∽ Cloud Code	minitube		Ln 28, Col 14 Spaces: 4 UTF-8 CRL	F Python R ⁴ O

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

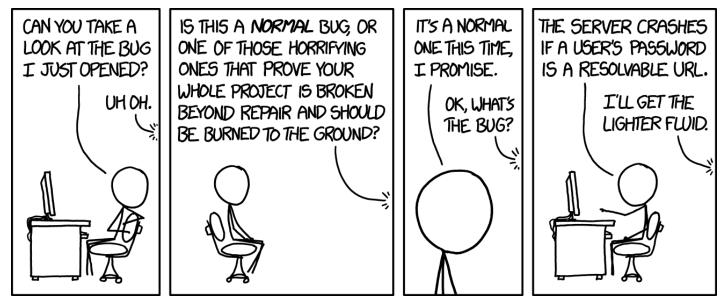




Back to these ML specific bugs

Or better know as

 $\underline{\land}$ When everything is running, but results are wrong $\underline{\land}$



Finding these buggers comes with experience

A potpourri of my findings over the last couple of years and <u>others</u>.

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:

- $\ensuremath{\textcircled{}^\circ}$ Seed everything and use same seed everywhere
- $\ensuremath{\textcircled{}^{\ensuremath{\mathbb{P}}}}$ Remove all data augmentation
- $\ensuremath{\mathbb{Q}}$ Use only a single batch of data where you have a feeling of the outcome

int getRandomNumber() { return 4; // chosen by fair dice roll. // guaranteed to be random. }

4. Investigate the math

Debug the math:

 $\ensuremath{\widehat{\mathbb{P}}}$ Go through your code, line by line

 \mathbb{P} There should be a one-to-one match between equations and lines of code

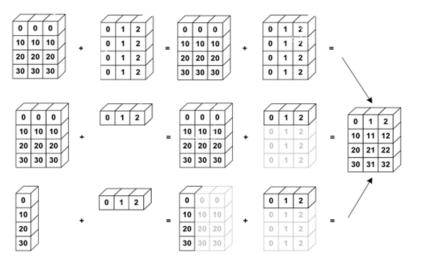
 $\ensuremath{\mathbb{Q}}$ Refactor if it is not clear

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)

- \mathbb{Q} Lookout for broadcasting!
- $\ensuremath{\textcircled{}^\circ}$ Broadcasting in python is both a blessing and a curse
- \mathbb{Q} 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()

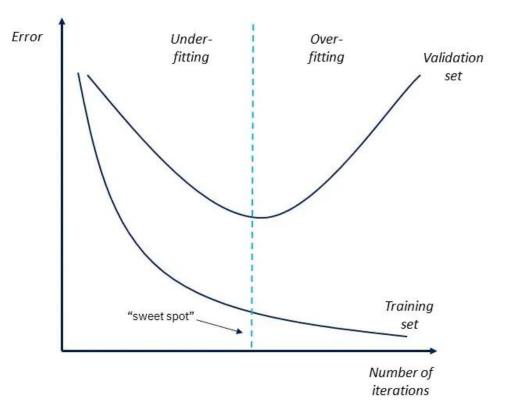
What is the problem here?

5. Overfitting is good?

Overfitting is usually seen as an bag thing bug...

 $\ensuremath{\textcircled{}^{\ensuremath{\otimes}}}$ Models should be able to memorize one batch

 $\[mathbb{Q}\]$ 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

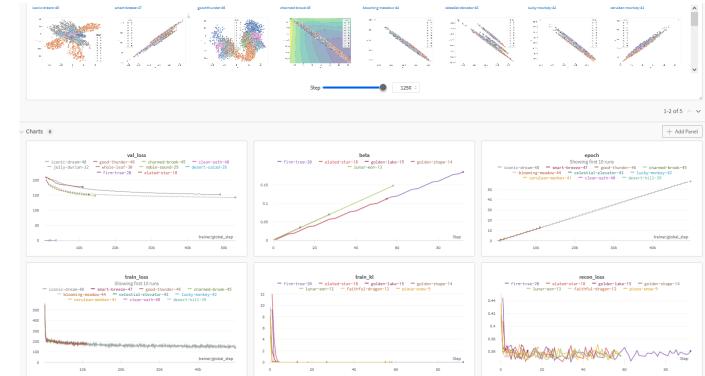
Log and visualize everything

Paining loss is really decreasing right?

 $\ensuremath{\textcircled{}^{\ensuremath{\wp}}}$ Can you add additional metrics?

Log dynamic changing
 hyperparameters (learning rate with
 Ir schedulers)

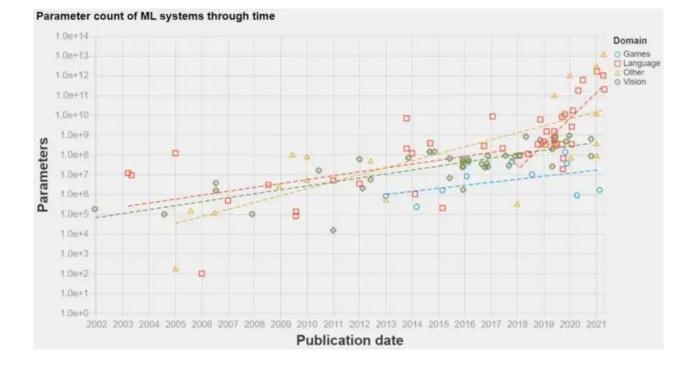
Plot data, predictions, reconstructions etc. over time



8. Add complexity over time

If you are still good, then

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

