11-785: Introduction to Deep Learning

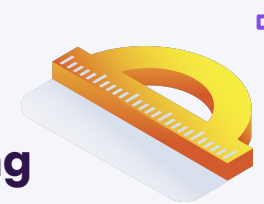
Recitation OL : Debugging Deep Neural Networks

Fall 2023

Shikhar Agnihotri
Harshith Arun Kumar

01 Common Debugging Scenarios

02 General Coding Tips



03 Visualizing image & speech Datasets

04 Debugging with breakpoints

Common Debugging Scenarios

Confused ?

“My code runs, but the accuracy is terribly low and not improving”

Blocked ?

“My code throws an error and stops running, and I don’t understand the lengthy error message”

Slow ?

”My model is taking forever to train.”

Consolidate Hyperparameters

Putting everything in one place helps with model experimentation. We can save everything in a config dictionary.

```
config = {
    "architecture": "convnext-t",
    "optimizer": "AdamW",
    "lr": 1e-4,
    "loss": "cross entropy",
    "scheduler": "reduce on plateau",
    "augmentations": "AutoAugment",
    "weight_decay": 0.05,
    "label smoothing": 0.1,
    "stochastic depth": 0.0,
    "regularization": "",
    "batch_size": 64,
    "epochs": 100,
    "label_smoothing": 0.2,
    "momentum": 0.9
}
```

Write test cases

Write a small test case for each function you write.

- Print the type and shape of important variables
- Slice and print only a segment of high dimensional variables
- Visualize the data using matplotlib, we will have a short code demo at the end of this recitation
- Break and print after one iteration

Use print statements

Control print statements via a debug flag

A switch between:

- Debugging mode: you want to print extra information
- Training mode: you don’t want to do anything redundant

```
if debug_flag==True:
    print("debugging information")
```

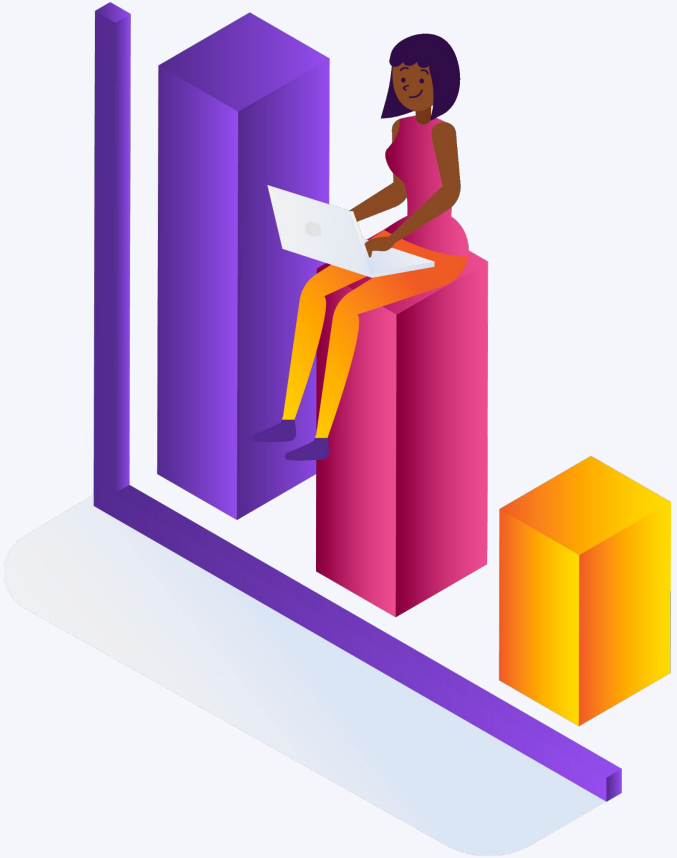
Restart Kernels

Control print statements via a debug flag

- If you are running your code in Jupyter Notebook or Google Colab, sometimes you accidentally ran a cell twice or you ran cells in the wrong order.
- Technically there is nothing wrong with your code – just restart kernel and rerun everything from the beginning!

Visualizing Image and Speech Datasets

Debugging via breakpoints



Types of Errors

- Error Type 1: Coding Error
  - Syntax Errors
  - Logic/Math Errors
  - Runtime Errors
- Error Type 2: Time Issue
- Error Type 3: Memory Issue
  - CUDA out of memory
  - Location of variables

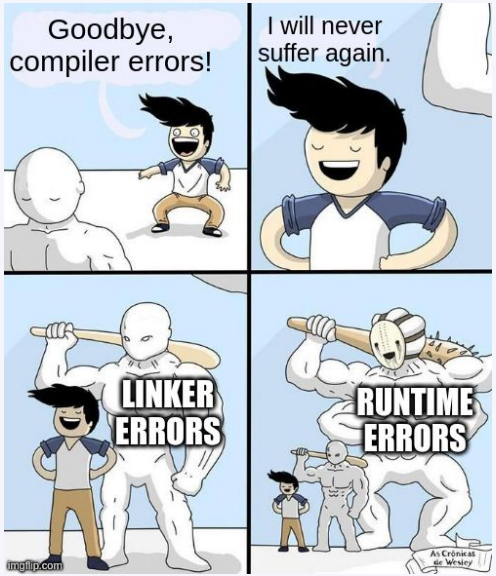
Error Type 1:

- **Syntax Errors:**
  - Stack Overflow → Best
  - Refer to Recitation 0A - Python and OOP fundamentals

Error Type 1:

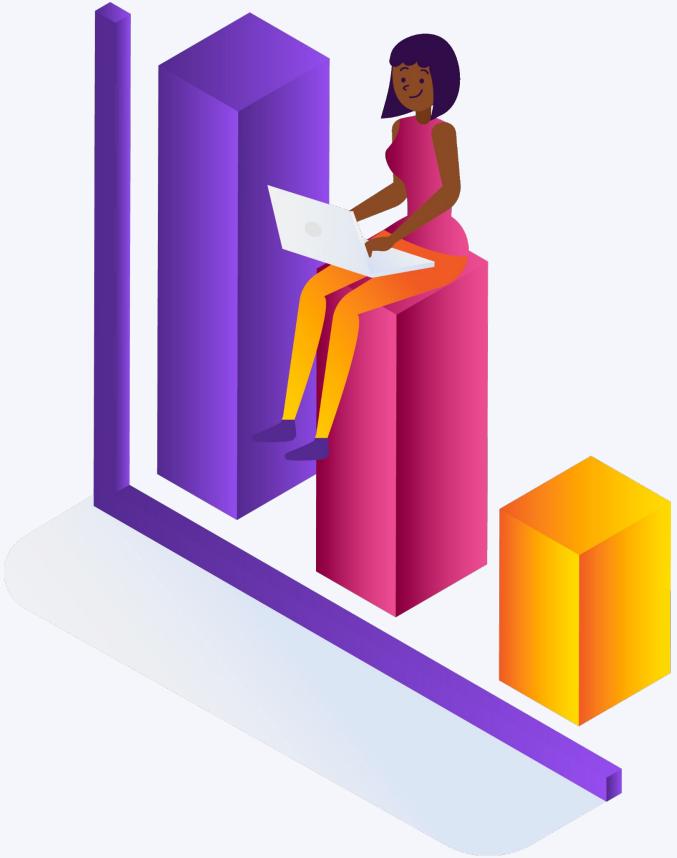
- Logic/Math Errors:
    - Lot of matrix multiplication and math (HWP1)
    - Read write-up
    - Check shapes of variables (Dimensions)
    - Read numpy documentation for a function for its:
        - Purpose
        - Input type, shape
        - Output type, shape

Error Type 1:



- **Runtime Errors**
  - Read traceback to find root of error
  - Read library documentation for function specifics
  - Set batch size to 1 and run the code on CPU - More readable error messages
- **Pdb**
  - Learn to use pdb → Interactive python debugger
  - Stack overflow → Best

Pdb demo



Error Type 2:

- **Time Issue**
  - I debugged all the syntax errors and my model runs
  - But takes 40 minutes to train an epoch
  - Ideally it is supposed to take 10 minute

Error Type 2:

- **Things to check**
  - If using GPU
  - Batch size (32 to 128, as large as your GPU does not complain)
  - Check data-loader and training loop:
    - Most iterations happen here
  - Use mixed_precision while training
  - Use time module to identify which part of the code is taking long

Error Type 3:

- **Memory Issue**
  - Model trains normally
  - But after 30 epochs:

Error Type 3:

- **Common errors:**
  - If you put too many things on GPU, you will see this:

```
RuntimeError: CUDA out of memory.
```

  - **Things to try:**
    - Reduce batch size
    - Use cuda mixed precision
    - Read Tutorial before starting on HW P2s:
      - https://pytorch.org/tutorials/recipes/recipes/amp_recipe.html

Error Type 3:

- **Common errors:**
  - If you put too many things on GPU, you will see this:

```
RuntimeError: CUDA out of memory.
```

  - **Things to try:**
    - Check if you used torch.inference mode() during validation and testing:
      - Disables gradient calculation, only needed for backward-prop during training
      - Reduces memory consumption
    - Call torch.cuda.empty cache() help reduce fragmentation of GPU memory in certain cases.

Error Type 3:

- **Common errors:**
  - Forgetting to move data to GPU for training, validation and testing of the model.

```
RuntimeError: Expected object of device type
cuda but got device type cpu
```

Error Type 3:

- **Common errors:**
  - In order to train a model on the GPU it is first necessary to send the model itself to the GPU:

```
device = "cuda" # GPU
model = model.to(device=device)
```

  - The second requirement for running the training loop on the GPU is to move the training data:

```
x, label = x.to(device), label.to(device)
```

Error Type 3:

- **Common errors:**
  - If you are not careful, there might be a mismatch between the locations of different data being used in a function.
  - Things to try:
  - You can find out which device your tensor data are on at different points in the code by using the device property:

```
print(x_train.device)
```

  - To move the data to CPU or to GPU:

```
x = x.to(device="cpu")  # move to CPU
x = x.to(device="cuda") # move to GPU
```