11-485/685/785, Fall 2024

Lab 03: Debugging Deep Learning Networks

TAs: Ketan Chaudhary, Alexander Moker, Khushali Daga

*** This recitation builds upon “Recitation 0N: Debugging” and provides a comprehensive overview of common scenarios, as well as a bank of debugging tools.**

Debugging deep learning networks

In Computer Science, debugging is always a big, painful part of the work.

In Deep Learning, it's even bigger and more painful because of multiple sources of errors.

- **Implementation bugs**
- **Dataset construction**
- **Data/model fit**
- **Hyperparameter choice**

Neural net training fails silently...

Goal of this lab:

Learn a skill of debugging deep learning networks on your own 🚀

Agenda

1. **How to debug HW Parts 1**

    a. VS Code Debugger (setting breakpoints)
    b. Interactive Python Debugger (pdb)

2. **How to debug HW Parts 2**

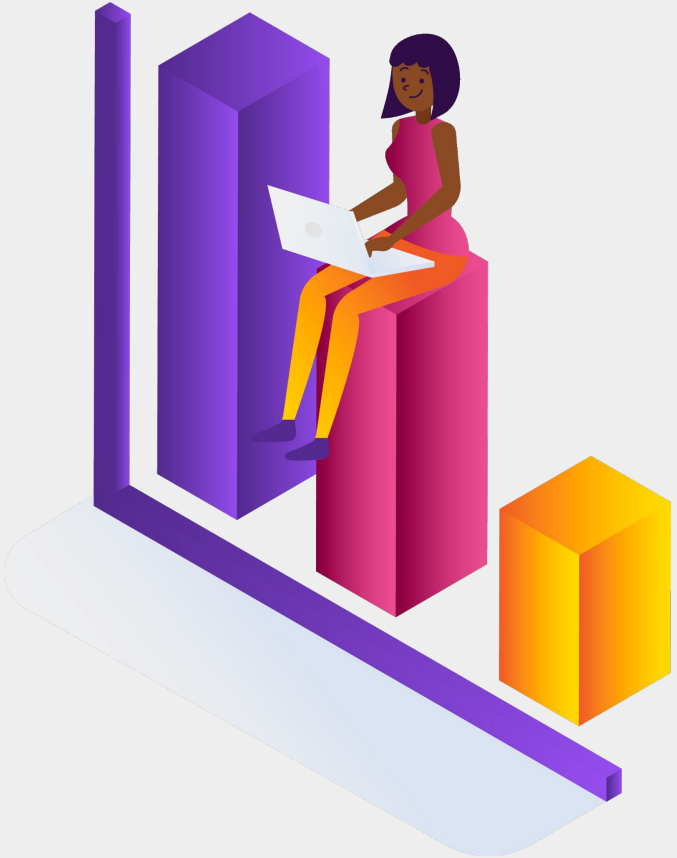
    a. Data loading and preprocessing
        i. EDA and Viz tools
    b. Building a model
        i. Hyperparameters consolidation
        ii. Checking dimensions and layers
    c. Training and monitoring
        i. Optimizing training time
        ii. Optimizing memory
        iii. Interpreting training performance
    d. Testing and Kaggle submission

How to debug HW Parts 1

VS Code Debugger – Live Demo

How to debug HW Parts 2

Pdb – Live Demo



HW Parts 2 ideal workflow

Step 0: Download the notebook.

Step 1: Complete all #TODOs and ensure your code runs and reaches very low cutoff.

Step 2: Divide the experiments among the study group members to achieve the high cutoff.

HW Parts 2 components

1: Data loading and preprocessing

2: Building a model

3: Training and monitoring

4: Testing and Kaggle submission

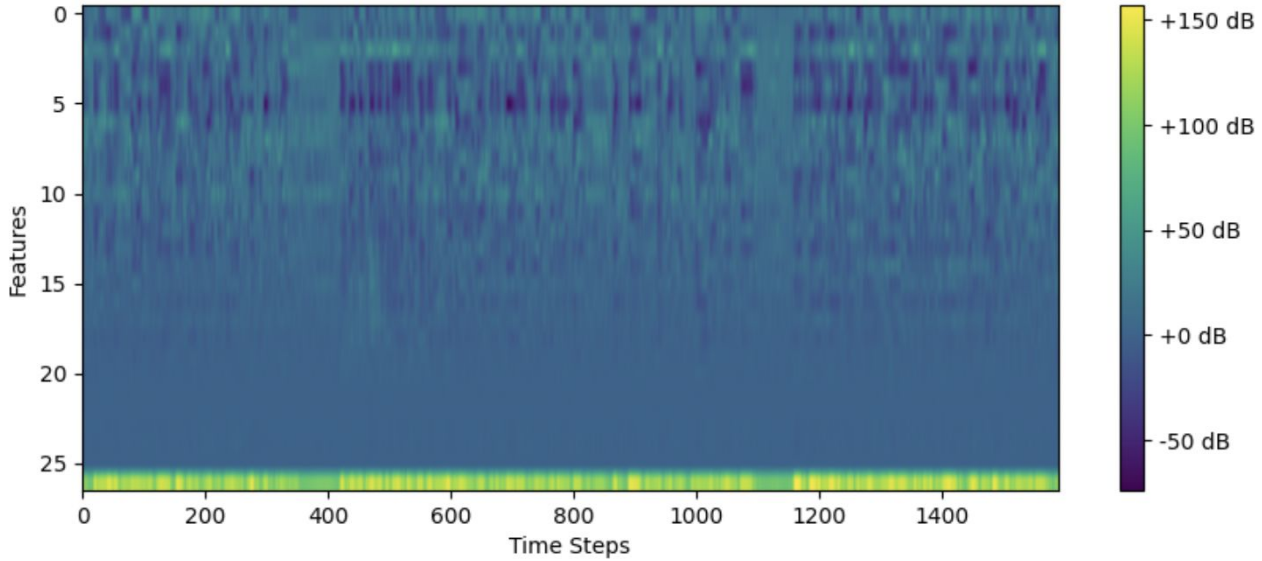
How to debug each part?

1. Data Loading and Preprocessing

#1.1 Taking a moment to review a single file from the dataset

Question 1: What's in our data? What are the inputs (x) and targets (y)? What should we do with our data to achieve great modeling results?

Single MFCC file:



Single transcript file:

```
['[SOS]' '[SIL]' '[SIL]' '[SIL]' '[SIL]' '[SIL]'] ['[SIL]' '[SIL]' '[SIL]' ... '[SIL]' '[SIL]' '[EOS]']
```

#1.2 Creating a train dataset class – mfcc → frames

Question 2: How to transform the original MFCC files into frames. How do their dimensions change?

Operation in the train/val dataset class	Resulting dimensions
1. Load the mfcc (audio) files	2 files (just a toy example) Each file is a matrix of size **10** (time steps) × **28** (features).
2. Normalize each mfcc file	No change
3. Concatenate all mfcc files into one	One matrix of size **20** (10+10) **× 28**
4. Pad the data with context of 10 at top & bottom	One matrix of size **40** (10 + 20 +10) **× 28**
5. Break the data into frames of size 2*context+1	Each frame is a matrix of size **21** (2 * context + 1) x **28** (features) Total number of frames is **20** (40 - 21 + 1)
6. Applies time/frequency masking to each frame	No change
7. Flatten and convert to a tensor	Each frame is converted from 2d into 1d with **588** (21*28) elements
8. Group frames into batches of size 5	**4 (20 // 5) batches** with 5 frames in each batch

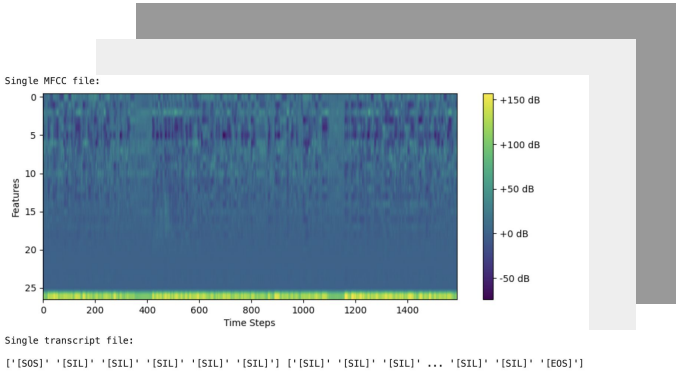
#1.2 Creating a train dataset class – transcripts → phonemes

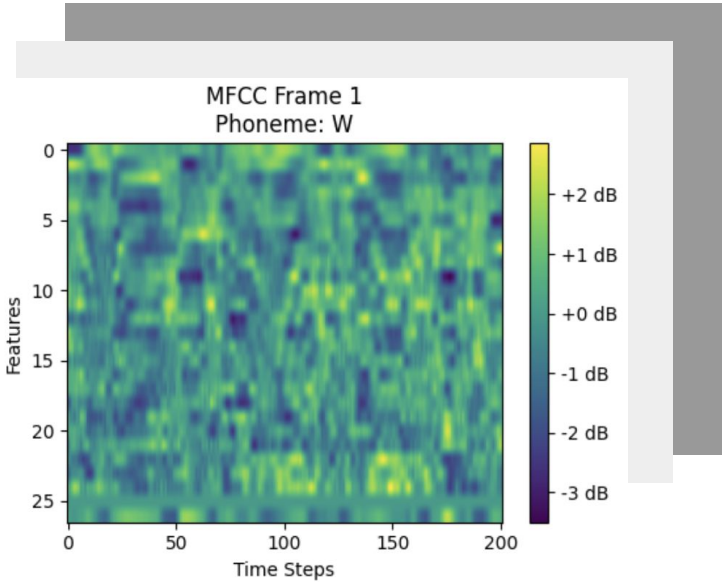
Question 3: How to transform the original transcripts into phonemes. How do their dimensions change?

Operation in the train/val dataset class	Resulting dimensions
1. Load the transcript files	2 files (just a toy example) Each file is of size **12** (time steps)
2. Remove [SOS] and [EOS] from each file	Each file is of size **10** (time steps)
3. Concatenate all files into one	One file of size **20** (10+10)
4. Convert the file into numerical format (phoneme-to-indices mapping) and convert to a tensor	No change
5. Group phonemes into batches of size 5	**4 (20 // 5) batches** with 5 phonemes in each batch

#1.2 Creating a train dataset class – one training sample

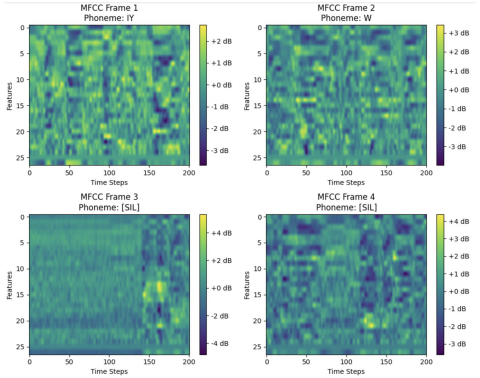
Question 4: What does the training sample look like after processing through the dataset/data loader?





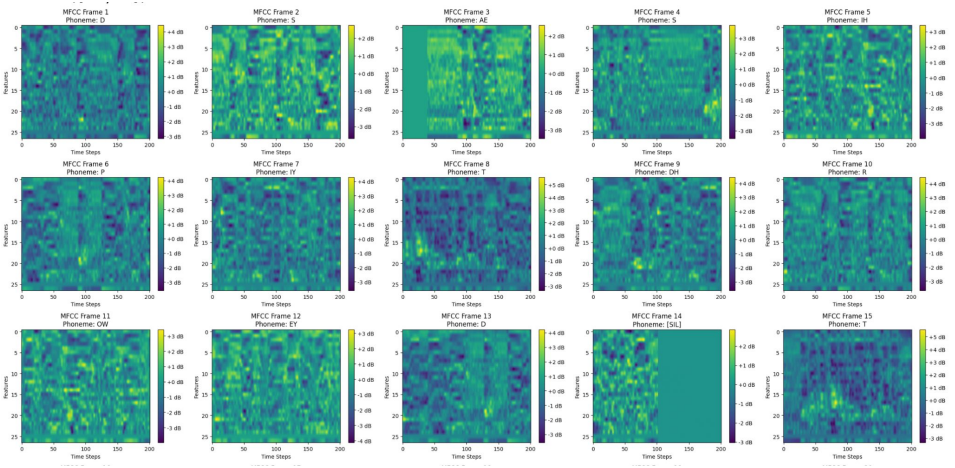
#1.2 Creating a train dataset class – one batch of training samples

Question 5: What does one batch look like? What is the difference between a batch size of 4 and 15, and how does batch size affect RAM memory usage?



Option A

Batch size = 4



Option B

Batch size = 15

#1.3 Creating a test dataset class

Question 6: What's the difference between train and test dataset classes?

Train/val dataset class

- MFCC (audio)
  - Sort the mfcc files in the directory
  - Load the mfcc files
  - Normalize the mfcc files
  - Concatenate all mfcc files into one
  - Pad the data with context
  - Break the data into frames
  - Apply time/frequency masking to each frame
  - Flatten each frame and convert to a tensor
  - Return a batch of frames
- Transcripts (phonemes)
  - Sort the transcript files in the directory
  - Load the transcripts
  - Remove [SOS] and [EOS] from each transcript
  - Concatenate all transcripts into one
  - Convert into numerical format
  - Convert each phoneme to a tensor
  - Return a batch of phonemes

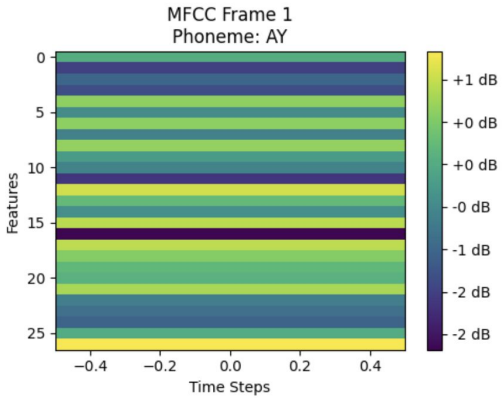
Test dataset class

- MFCC (audio)
  - Sort the mfcc files in the directory
  - Load the mfcc files
  - Normalize the mfcc files
  - Concatenate all mfcc files into one
  - Pad the data with context
  - Break the data into frames
  - [redacted]
  - Flatten each frame and convert to a tensor
  - Return a batch of frames

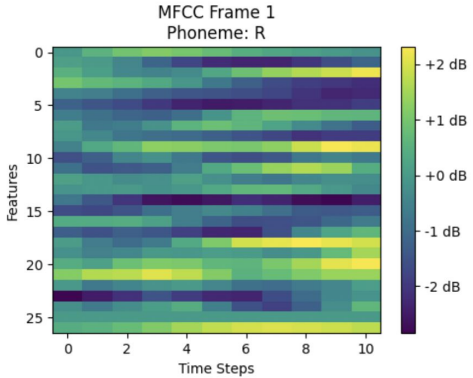
#1.4 Adding the context

Question 7: Is adding context useful? If so, what should the context size be?

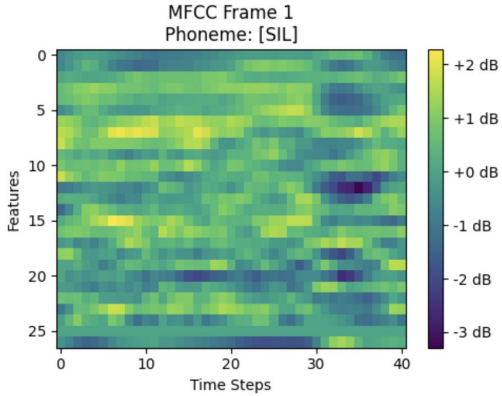
Can you guess which context size was applied in each of the following cases without looking at the answers?



Option A
Context = 0



Option B
Context = 5

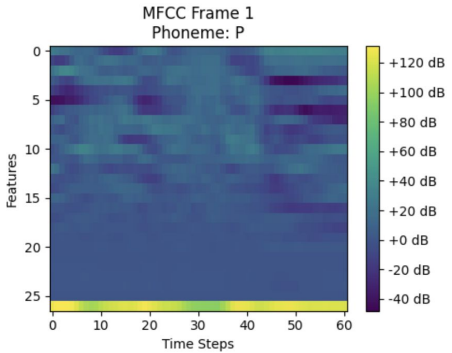


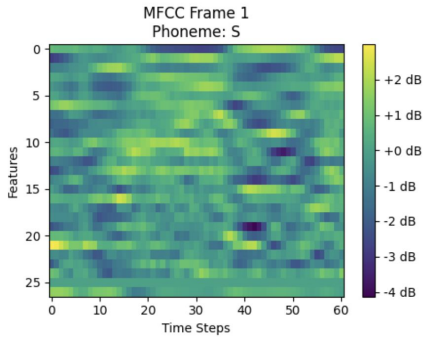
Option C
Context = 20

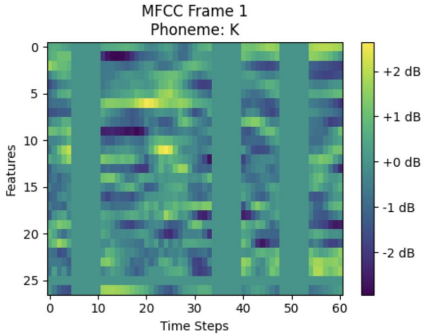
#1.5 Applying data normalization and transformations

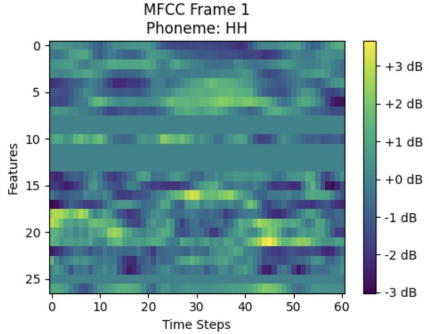
Question 8: Is normalizing and transforming the data helpful?

Can you guess which preprocessing methods were applied to the data below without looking at the answers?









Option A
Raw mfcc frame

Option B
Normalized mfcc frame

Option C
Normalized + time masking

Option D
Normalized + frequency masking

#1.6 Selecting number of workers in dataloaders

Common errors: Using too many “workers” - subprocesses responsible for loading and preprocessing data

```
RuntimeError: DataLoader worker (pid(s) 1978) exited unexpectedly
```

Things to try:

- Try reducing the number of workers in the dataloader.

2. Building a Model

#2.1 Centralization of hyperparameters

Question 9: Which option is better: using hardcoded values or those stored in a config dictionary/yaml file?

Option A

```
INPUT_SIZE  = (2 * 35 + 1) * 28
OUTPUT_SIZE = 42
model       = Network(INPUT_SIZE, OUTPUT_SIZE).to(device)
summary(model, frames.to(device))
```

Option B

```
INPUT_SIZE  = (2*config['context'] + 1) * config['mfcc_features']
OUTPUT_SIZE = config['vocab_size']
model       = Network(INPUT_SIZE, OUTPUT_SIZE).to(device)
summary(model, frames.to(device))
```

#2.1 Centralization of hyperparameters

Approach 1 - config dictionary

```
[42] config = {
              # Dataset ----
              "batch_size": 10,
              "mfcc_features": 27,
              "vocab_size": 42,
              "context": 35,

              # Model ----
              "dropout": 0.25,

              # Training ----
              "learning_rate": 0.001,
              "epochs": 50,
              }
```

```
[44] config['batch_size'] # call the hyperparameter

     10
```

Approach 2 - config.yaml file

```
%%writefile config.yaml

# Dataset ----
batch_size: 10
mfcc_features: 27
vocab_size: 42
context: 35

# Model ----
dropout: 0.25

# Training ----
learning_rate: 0.001
epochs: 50
```

```
[34] with open("config.yaml") as file:
         config = yaml.safe_load(file)
```

```
[35] config['batch_size'] # call the hyperparameter

     10
```

#2.2 Reading the model summary table

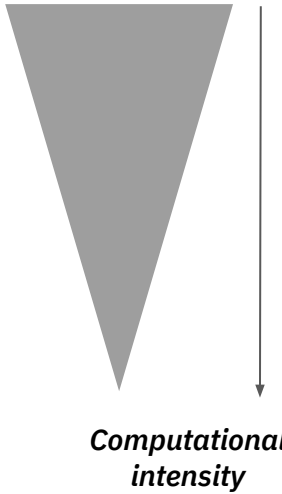
Question 10: How many parameters does this model have? Which layer is the most computationally intense?

Layer	Kernel Shape	Output Shape	Params	Mult-Adds
0_model.Linear_0	[5427, 512]	[100, 512]	2.779136M	2.778624M
1_model.BatchNorm1d_1	[512]	[100, 512]	1.024k	512.0
2_model.ReLU_2	-	[100, 512]	-	-
3_model.Linear_3	[512, 1024]	[100, 1024]	525.312k	524.288k
4_model.BatchNorm1d_4	[1024]	[100, 1024]	2.048k	1.024k
5_model.ReLU_5	-	[100, 1024]	-	-
6_model.Linear_6	[1024, 512]	[100, 512]	524.8k	524.288k
7_model.BatchNorm1d_7	[512]	[100, 512]	1.024k	512.0
8_model.ReLU_8	-	[100, 512]	-	-
9_model.Linear_9	[512, 42]	[100, 42]	21.546k	21.504k

Totals	
Total params	3.85489M
Trainable params	3.85489M
Non-trainable params	0.0
Mult-Adds	3.850752M

The most computationally intense layer

Total number of parameters



Computational intensity

#2.3 Increasing the number of model parameters - going deep/wide

Question 11: The student has decided to increase the number of layers but something went wrong. What's the issue?

```python
class Network(torch.nn.Module):

    def __init__(self, input_size, output_size):

        super(Network, self).__init__()
        self.model = torch.nn.Sequential(

            # layer 1
            torch.nn.Linear(input_size, 512),
            torch.nn.ReLU(),

            # layer 2
            torch.nn.Linear(1024, 1024),
            torch.nn.ReLU(),

            # layer 3
            torch.nn.Linear(1024, 512),
            torch.nn.ReLU(),

            # output layer
            torch.nn.Linear(512, output_size))

    def forward(self, x):
        out = self.model(x)
        return out
```

```
RuntimeError: mat1 and mat2 shapes cannot be multiplied (100x512 and 1024x1024)
```

Wrong input shape

#2.4 Introducing batch normalization

Question 12: The student has decided to add batch normalization but something went wrong. What's the issue?

```
class Network(torch.nn.Module):

    def __init__(self, input_size, output_size):

        super(Network, self).__init__()
        self.model = torch.nn.Sequential(

            # layer 1
            torch.nn.Linear(input_size, 512),
            torch.nn.BatchNorm1d(512),
            torch.nn.ReLU(),

            # layer 2
            torch.nn.Linear(512, 1024),
            torch.nn.BatchNorm1d(512),
            torch.nn.ReLU(),

            # output layer
            torch.nn.Linear(512, output_size))

    def forward(self, x):
        out = self.model(x)
        return out
```

```
RuntimeError: running_mean should contain 1024 elements not 512
```

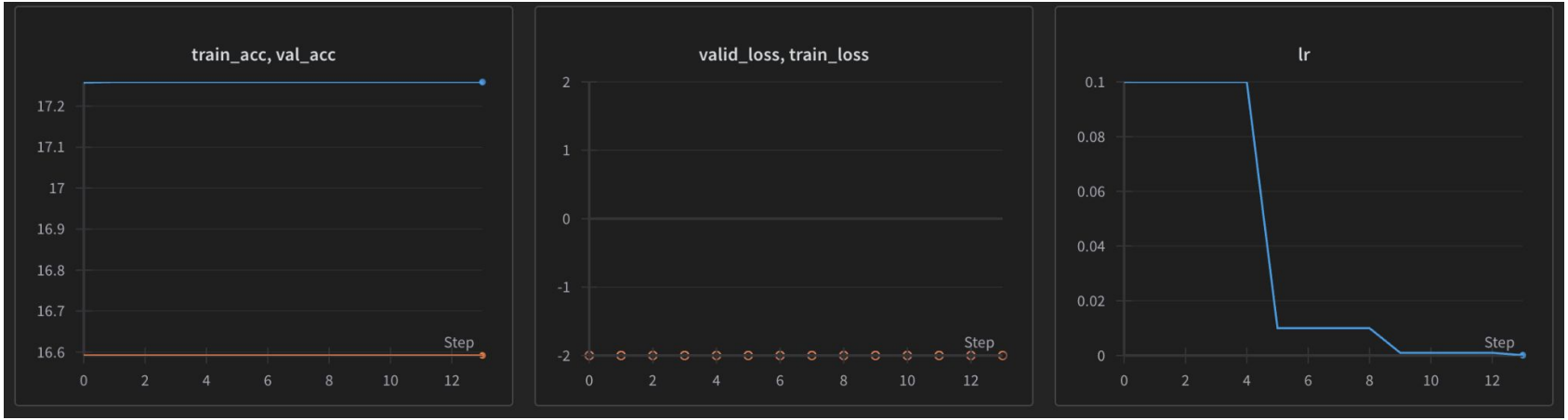
Wrong number of elements for batch normalization

3. Training and Monitoring

3.1 Interpreting Model Performance

Model performance interpretation

Question 13: Any issues? Should we stop training?

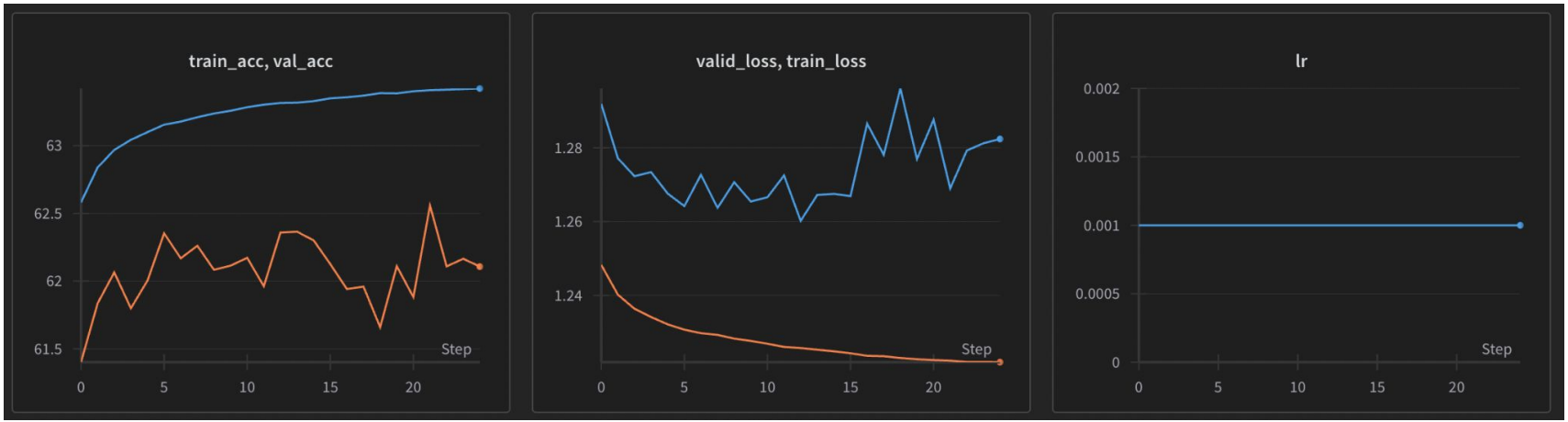


Legend: Blue lines - train; Orange - validation

Answer: Train and validation losses go to NAN → stop training, lower the learning rate, implement gradient clipping

Model performance interpretation

Question 14: Any issues? Should we stop training?

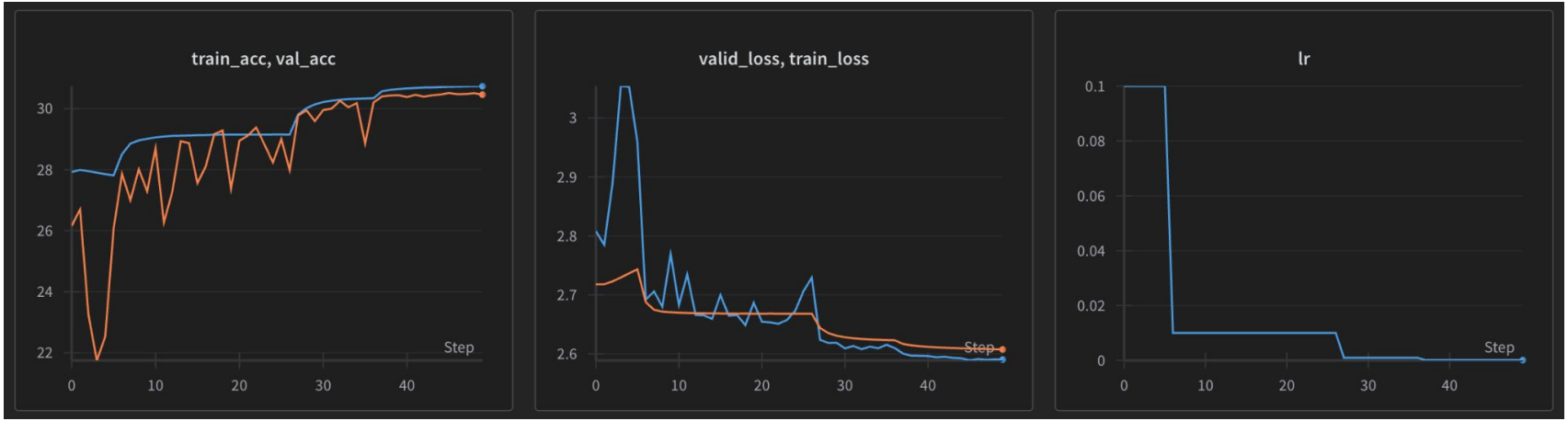


Legend: Blue lines - train; Orange - validation

Answer: Model doesn't converge → stop training, try implementing the learning rate scheduler to adjust the learning rate dynamically during training so the model won't stuck in local minima

Model performance interpretation

Question 15: Any issues? Should we stop training?

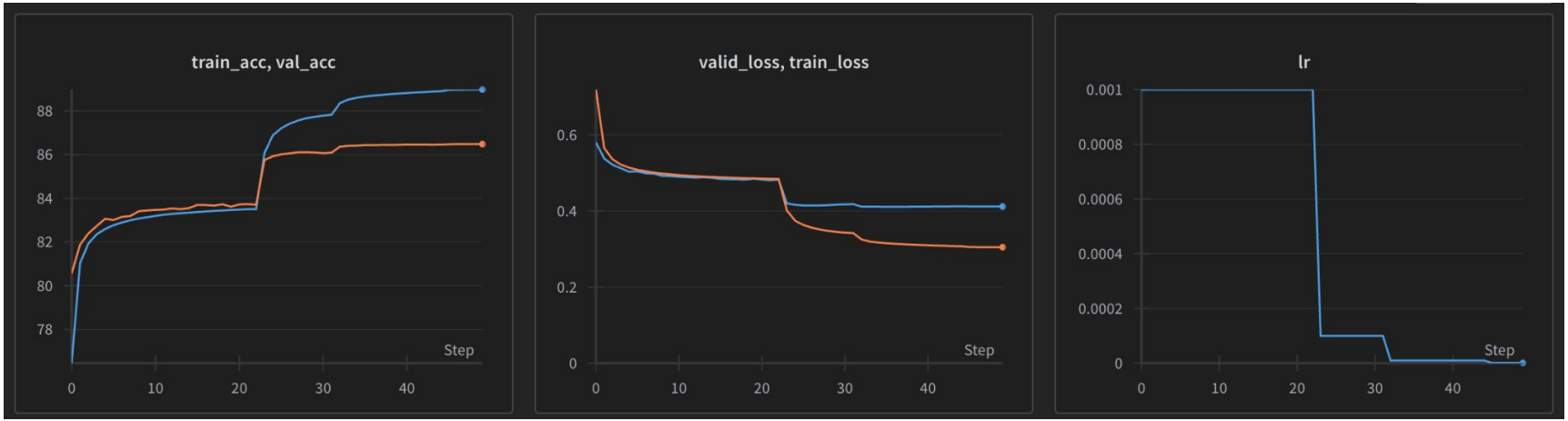


Legend: Blue lines - train; orange - validation

Answer: The model converges, but the performance is not great (accuracy is ~30% after ~50 epochs) → stop training, consider increasing the model's complexity (i.e., adding more parameters).

Model performance interpretation

Question 16: Any issues? Should we stop training?

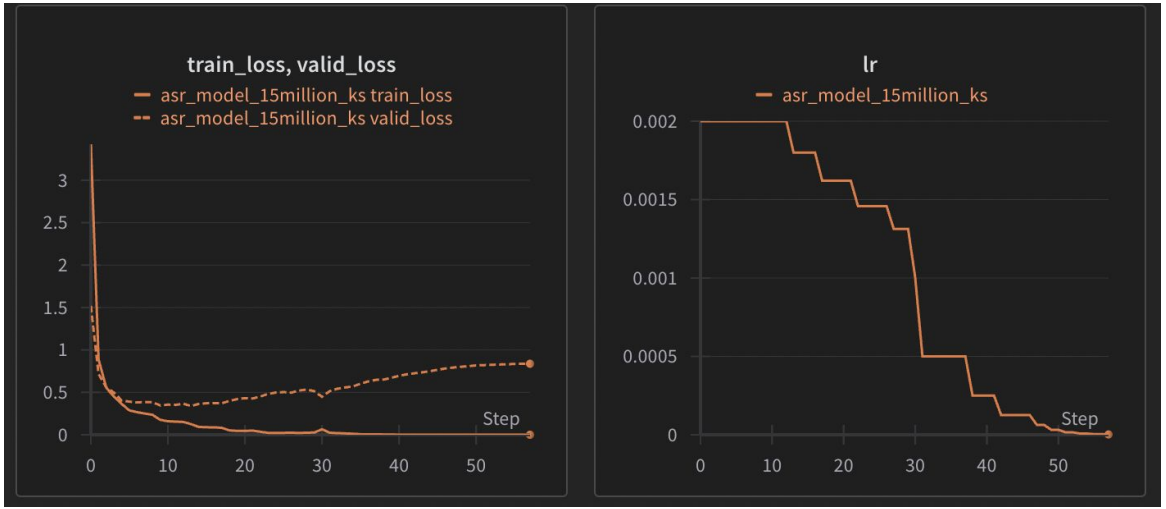


Legend: Blue lines - train; Orange - validation

Answer: The model starts to overfit after ~epoch 23 (valid loss no longer decreases) → stop training and consider regularizations

Model performance interpretation

Question 17: Any issues? Should we stop training?

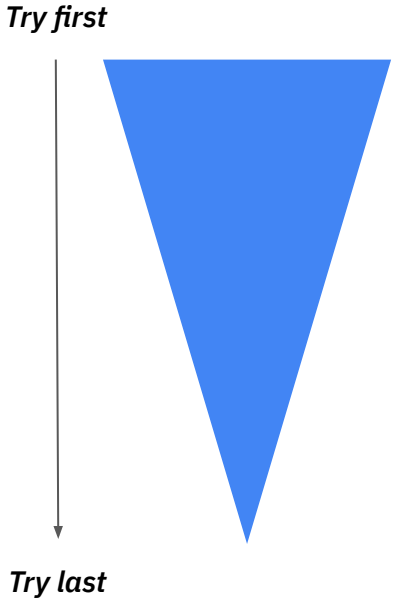


Legend: Solid lines - train; Dotted lines - validation

Answer: Classic overfitting (validation loss increases) → top training and consider regularizations

Addressing overfitting

Question 19: Which techniques have you already used in your homework?

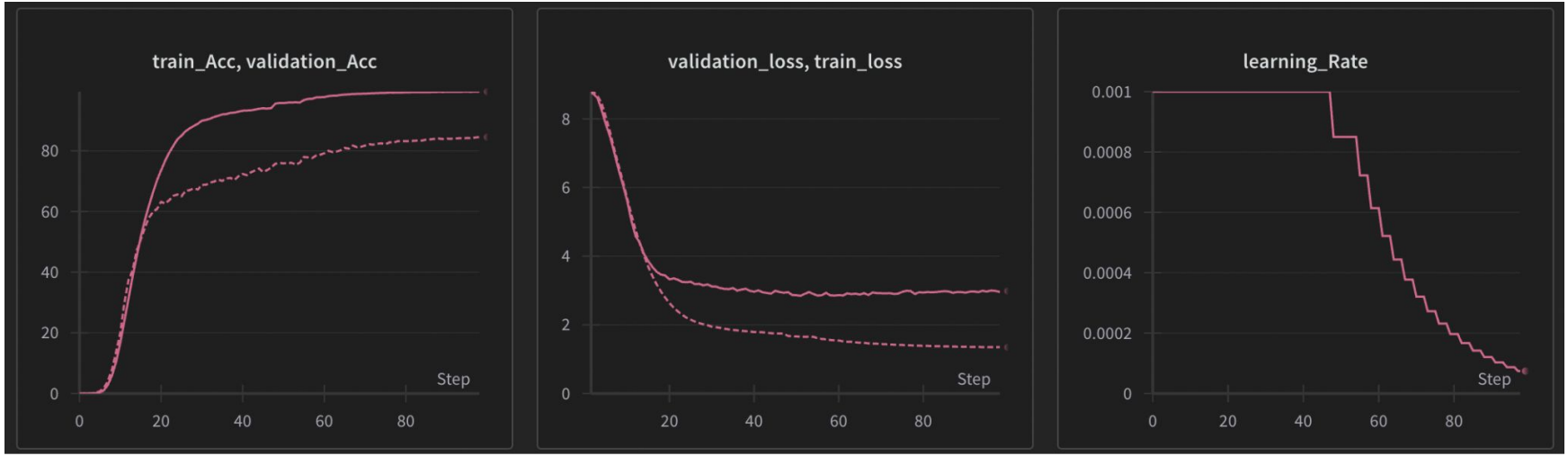


1. Add **more data** (if possible)
2. Add **normalization** (cepstral, batch norm, layer norm)
3. Add **data augmentation/transformation**
4. Increase **regularization** (dropout, weight decay)
5. **Error analysis**
6. **Different model architectures**
7. **Tune hyperparameters** (manual or grid search)
8. **Early stopping**
9. **Remove features**

https://fullstackdeeplearning.com/spring2021/lecture-7/

Model performance interpretation

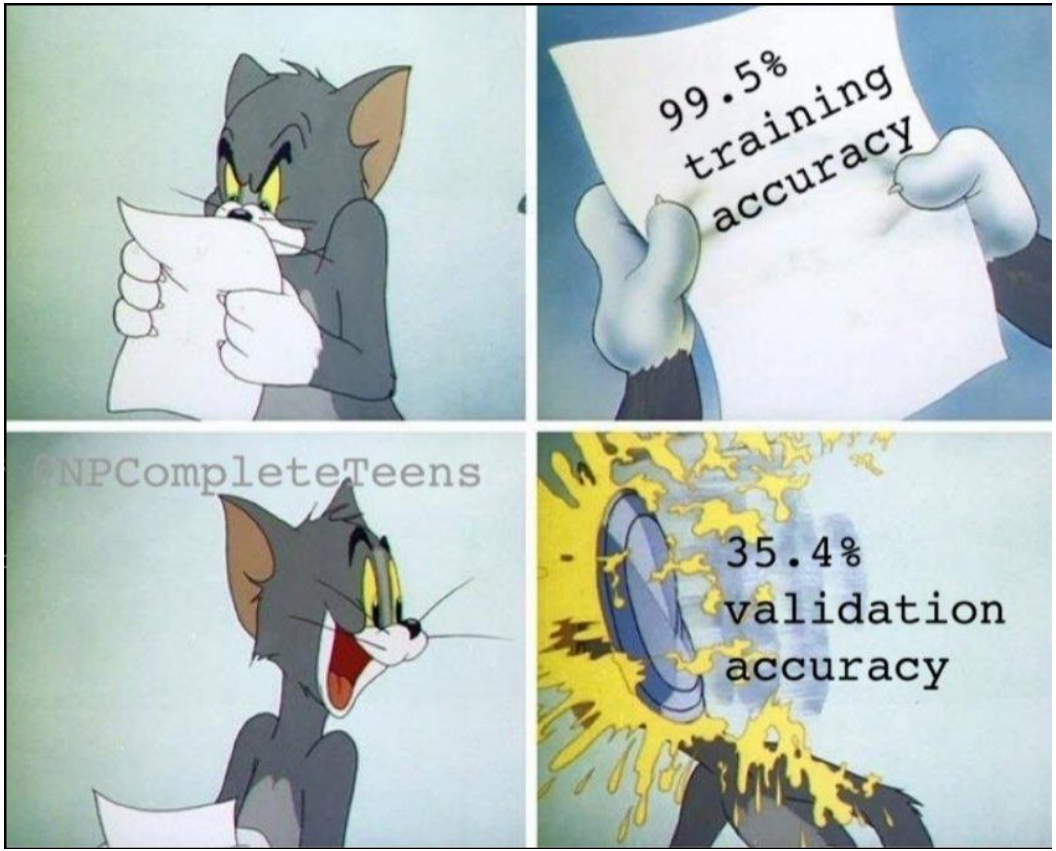
Question 18: Any issues? Should we stop training?



Legend: Solid lines - train; Dotted lines - validation

Answer: Valid accuracy continues to increase and valid loss continues to decrease → continue training

Classic Overfitting



3.2 Runtime issues

- 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
- Learn to use pdb → Interactive python debugger
- Stack overflow → Best

3.3 Memory issues

- Model trains normally
- But after 30 epochs

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 → refer to this tutorial
- 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.

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

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

Things to try:

- In order to train a model on the GPU → send the model and data itself to the GPU

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

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

3.4 Time issues

- 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

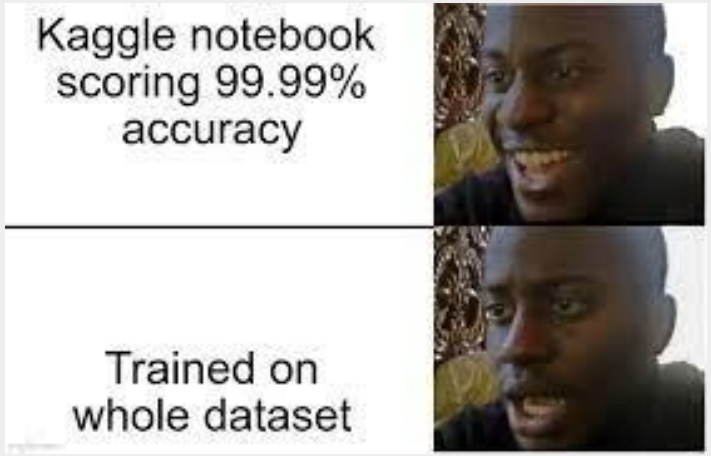
Things to check:

- 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

4. Testing and Kaggle submission

4. Testing and Kaggle submission

4. Testing and Kaggle submission



Creating a submission file

Question 20: Given the toy dataset below, how many rows of predictions should be in the submission file?

```
Number of files:
train_dataset: 2
val_dataset: 2
test_dataset: 1

Length of the dataset after concatenating all files:
train_dataset: 2994
val_dataset: 1058
test_dataset: 1039
```

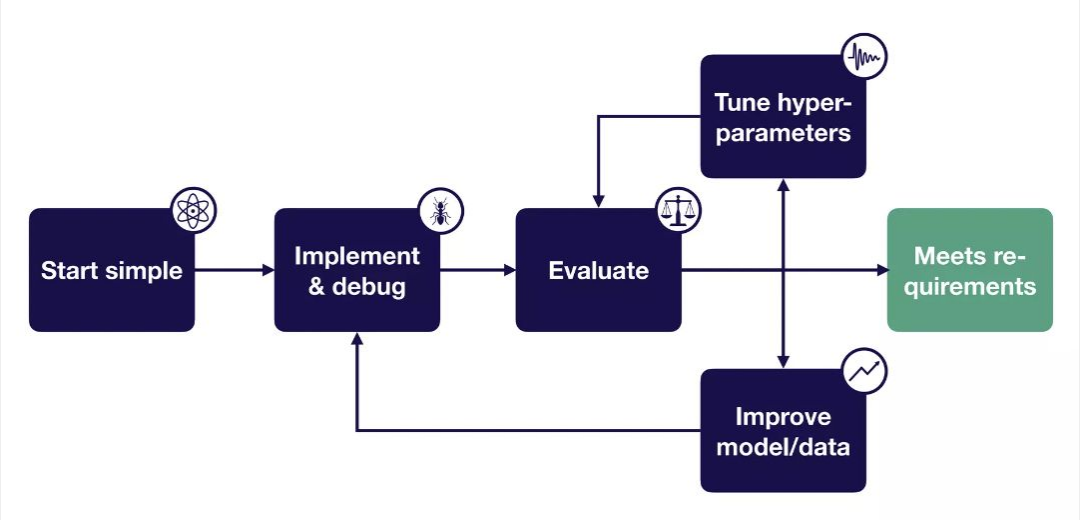
= number of rows in the submission file

sample_submission.csv (25.97 MB)

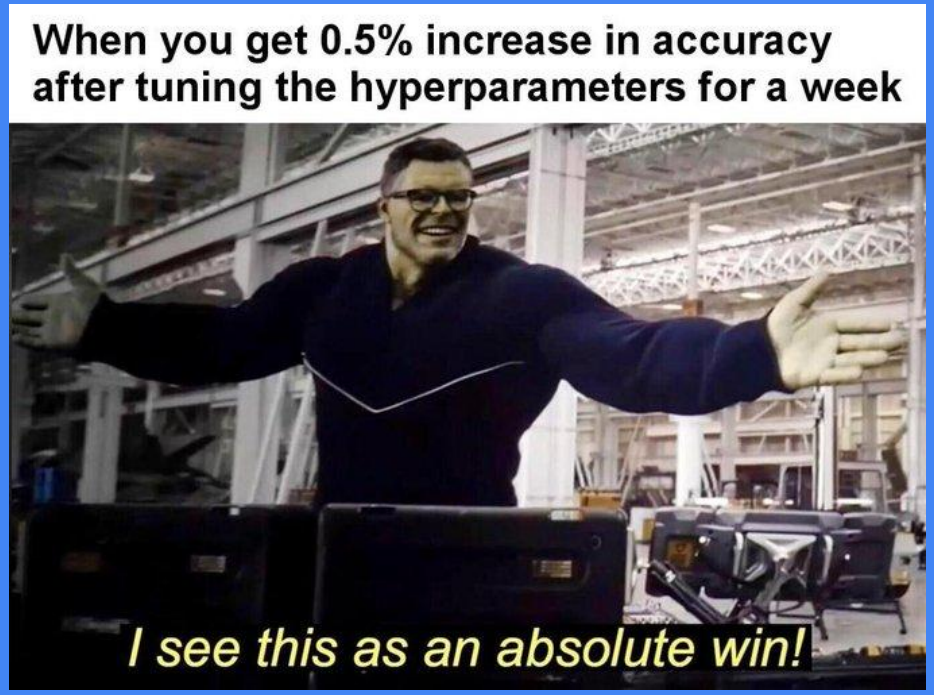
Detail | Compact | Column

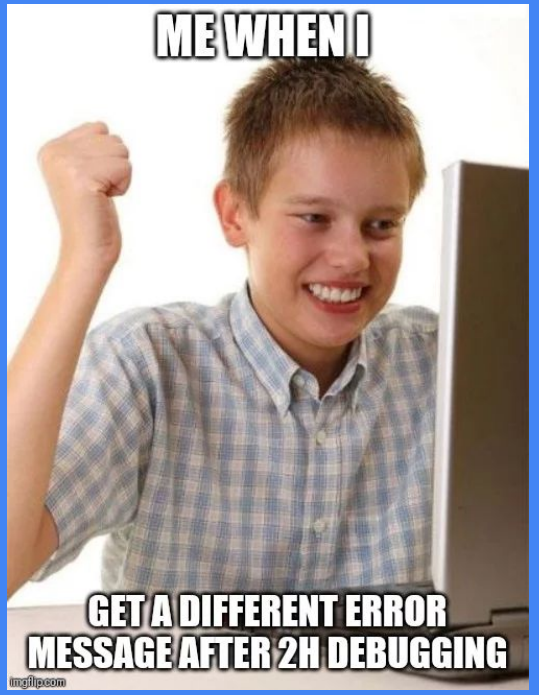
id	label
0 – 1.93m	1 unique value
0	[SIL]
1	[SIL]
2	[SIL]
3	[SIL]
4	[SIL]
5	[SIL]
6	[SIL]
7	[SIL]

General strategy for model debugging



https://fullstackdeeplearning.com/spring2021/lecture-7/





Thanks!

See you on Piazza and OHs!