MAS S60: PyTorch & Huggingface Tutorial

Before class:
Register for an Huggingface & Wandb Account
Open Colabs below:
Pytorch: tinyurl.com/s60torch
Huggingface: tinyurl.com/s60huf
Optional: View Slides at s60.dd.works

Huggingface Tutorial

- “Huggingface” is a set of multiple packages
  - transformers: Provides API to initialize large pretrained models
  - datasets: Provides easy way to download datasets
- Not from Huggingface but often used together
  - bitsandbytes: Provides functions to quantize large models
  - flash-attn: Allows the model to run faster with less memory

- Some terms to keep in mind
  - LoRA: Adapter to train large models with less memory
  - Bfloat16: Robust half precision representation often used to save memory

How to AI (Almost) Anything

Debugging Checklist

Special thanks to Andrej Karpathy and Pierce Freeman

mit media lab

MIT EECS

Why?

- Neural nets can fail silently – unlike regular software bugs
- Requires a structured process to avoid hidden errors
- Goals: Build confidence in your setup and achieve reliable results

See Ali Rahimi's Test of Time Blog

multisensory intelligence

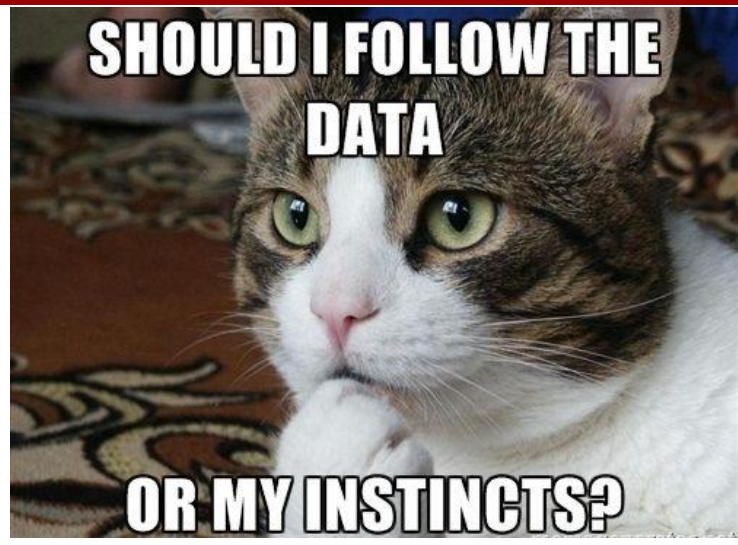
Some Observations

- Neural net training is a leaky abstraction
  - Not plug-and-play like typical software libraries
  - Complexity must be understood to avoid failure
- Neural net training fails silently
  - Logical errors, not syntactic ones
  - Networks can train but yield suboptimal results

The Recipe

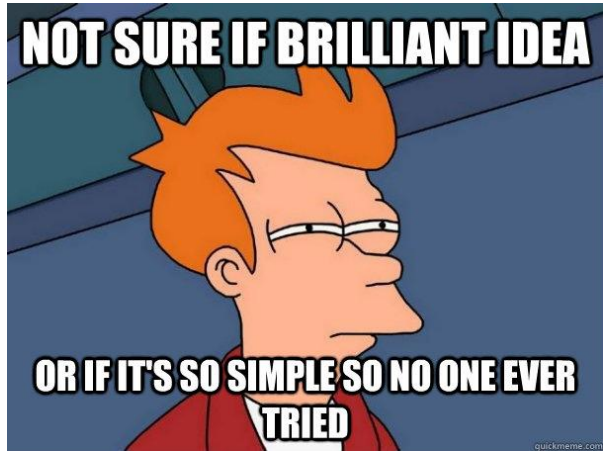
- Become one with the Data
- Set up end-to-end skeleton and get dumb baselines
- Overfit to diagnose errors
- Regularize for better generalization
- Tune hyperparameters
- Squeeze out final improvements

Becoming one with the Data



- Spend hours reviewing data samples
- Check for duplicates, corrupt data, and label errors
- Identify patterns, biases, and potential preprocessing steps
- Write code to filter/sort and visualize distributions

Set up Skeleton + Baselines

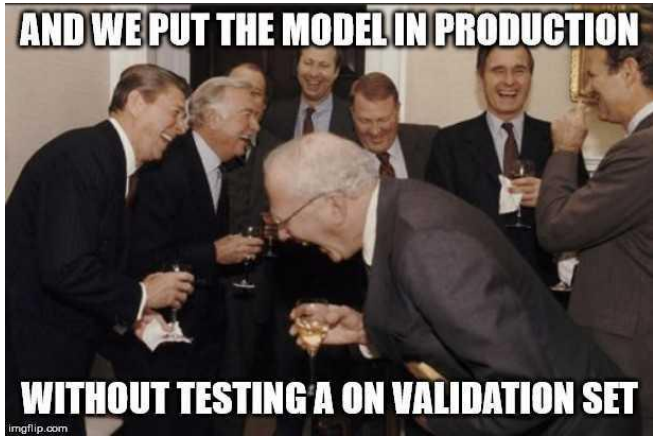


- Start with a simple model (linear classifier or tiny ConvNet)
- Tips & Tricks
  - Fix random seeds for reproducibility
  - Simplify by disabling augmentation and complex features
  - Verify loss at initialization
  - Overfit on one batch to confirm correctness

Overfit

- Use a simple model to overfit on a very small training set
- Debug if you cannot reach a low error rate
- Tips:
  - Visualize predictions and loss dynamics
  - Ensure gradients affect only expected inputs
  - Avoid premature optimization of architecture

Regularize for Generalization



- Add more real data – the best way to reduce overfitting
- Use data augmentation and pretraining
- Reduce input dimensions and model size
- Techniques: Dropout, weight decay, early stopping

Tune for Hyperparameters

- Prefer random search over grid search
- Use Bayesian optimization tools when available
- Don’t overcomplicate – start with simple models

Squeeze Out the Juice

- Use model ensembles for a 2% performance boost
- Leave models training longer than expected
- Explore state-of-the-art architectures and papers

Conclusion

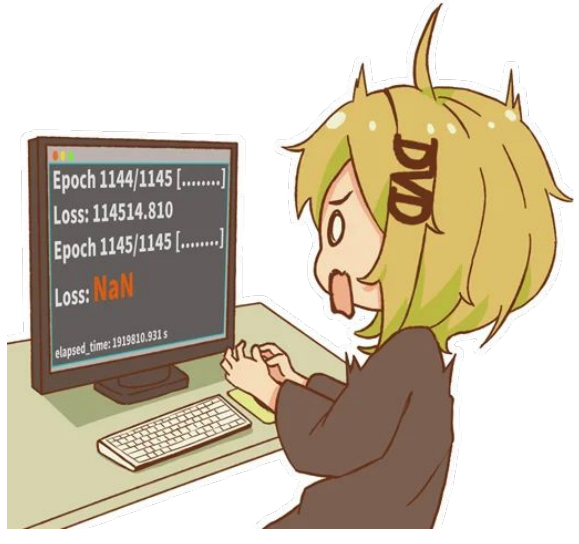
- Follow a structured approach: Data → Baselines → Overfit → Regularize → Tune
- Be patient and meticulous
- Visualize everything, hypothesize, and validate
- Ready to achieve SOTA results

How to Design ML Models for New Data

- Look at the data first
- For simple, low dimensional data, start with simple models (SVM, Random Forest, Shallow MLP/CNN)
- For vision/language data, try pretrained model
- Start simple, then add complexity. Simple ones can be used as baselines.

How to Debug Your Model

- **Look at the data first. Is the input data & label correct?**
  - Ensure no data leakage;
- **Look at the outputs. Is model only predicting one label?**
  - Label imbalance: Data Augmentation; loss scaling
- **Look at the training loss**
  - Loss is nan: Inspect weights and inputs for NaN values. Make sure weights are initialized. LLM: Use bfloat16 instead of float16.
  - Loss not changing: Model underfitting. Increase learning rate; decrease weight decay; Add more complexity; Use better optimizer*.



* Personal tip: I recommend trying second order optimizers from packages like Heavyball

How to Debug Your Model (Continued)

- Look at Loss (Continued)
  - Loss highly varied/increasing: Decrease learning rate; Gradient Clipping; Use better Optimizers
- Look at train vs val accuracy (or any other metrics)
  - Train >> Val: Model overfitting. More weight decay, reduce model complexity, data augmentation, get more data
  - Train ≈ Val ≈ 100%: Check for data leakage