Model Training

Andreas Marfurt
MLOps

Information Technology
25.09.2025

FH Zentralschweiz

Overview

- Experiment tracking
- Hyperparameter optimization
- Learning curves

Experiment tracking

- Why?
- How
  - What to track
  - How to track, Tools
  - Demos

Why experiment tracking?

- Running experiments with neural networks can get messy very quickly
  - Just a small code change/bug fix
  - Try a different learning rate
  - Update a library
  - Run again on a different machine
- It's easy to lose track of all the things that have been tried
- Want to find the best model, and then be able to also use it
  - Nothing feels worse than having a great result but not being able to reproduce it

→ Structure your project, follow a process, use tools!

Saving model checkpoints

- Save your models, so you can run them again later
- Different methods
  - Save parameters only vs. entire model

(recommended)

Save:

```
torch.save(model.state_dict(), PATH)
```

Load:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

(more closely tied to code implementation and file structure → breaks more easily)

Save:

```
torch.save(model, PATH)
```

Load:

```
# Model class must be defined somewhere
model = torch.load(PATH)
model.eval()
```

Saving model checkpoints

- Save model/optimizer checkpoints to resume training (PyTorch tutorial)

Save:

```
torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss,
            ...
            }, PATH)
```

Load:

```
model = TheModelClass(*args, **kwargs)
optimizer = TheOptimizerClass(*args, **kwargs)

checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['loss']

model.eval()
# - or -
model.train()
```

What to track

- Train/validation loss
- Other metrics (accuracy, F1, ROUGE, model-based scores, …)
  - More in the next lecture
- Speed
- Resource (CPU, GPU, memory, I/O) utilization
- Hyperparameters

→ Measure how this develops over the course of training

HSLU

Simplest form: Logging

- Log to file/stdout

```python
import logging
logging.basicConfig(
    filename='example.log',
    encoding='utf-8',
    level=logging.DEBUG,
    format="%(asctime)s :: %(name)s :: %(message)s"
)
logging.debug('This message should go to the log file')
logging.info('So should this')
logging.warning('And this, too')
logging.error('And non-ASCII stuff, too, like Øresund and Malmö')
```

example.log

```
2023-08-03 09:55:39,597 :: DEBUG :: This message should go to the log file
2023-08-03 09:55:39,597 :: INFO :: So should this
2023-08-03 09:55:39,597 :: WARNING :: And this, too
2023-08-03 09:55:39,597 :: ERROR :: And non-ASCII stuff, too, like Øresund and Malmö
2023-08-03 09:55:55,982 :: DEBUG :: This message should go to the log file
```

- Efficient and accurate, but not easy to compare different runs

Folder structure

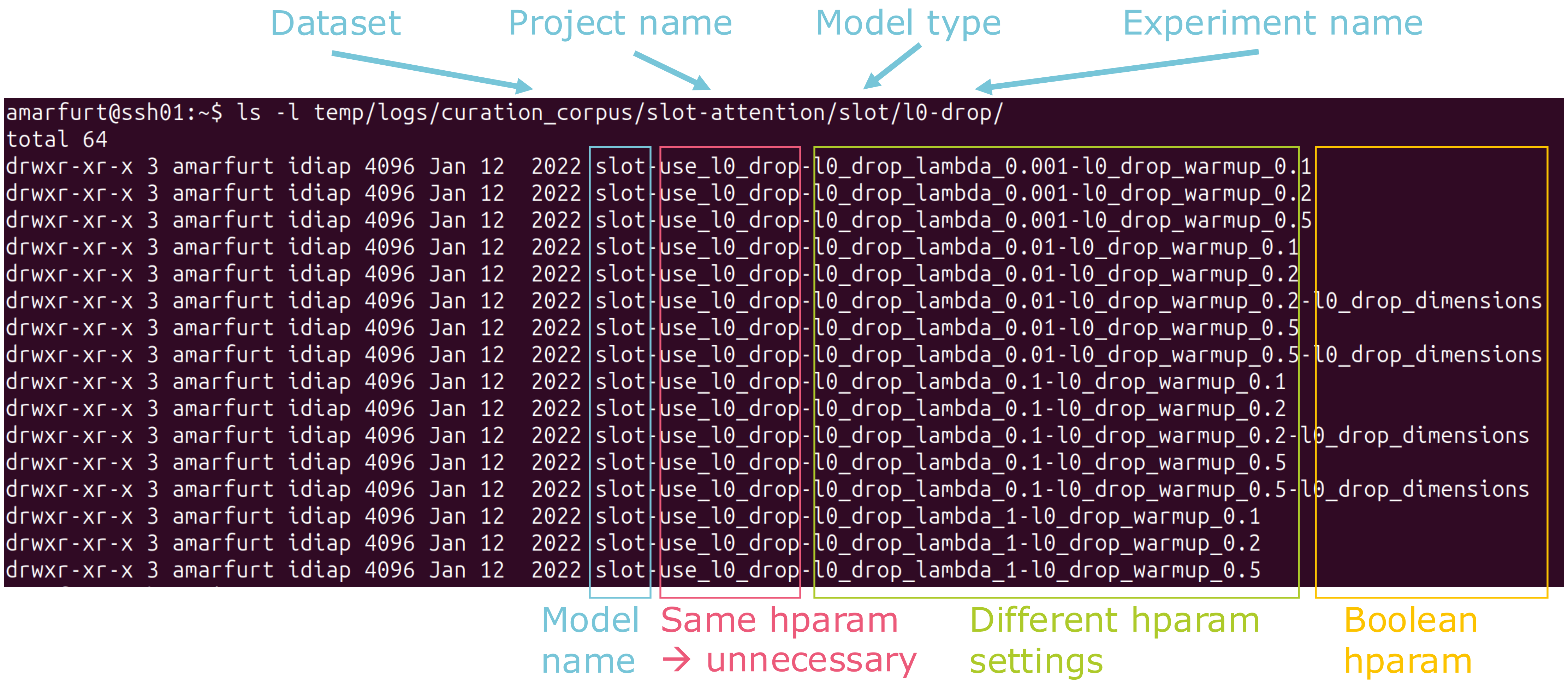
- Choose meaningful names
  - Not meaningful: v1, v2, v3
- Idea from my projects:

```
project_name/dataset/task/model_type/experiment_name
```

- Experiment name
  - Including hyperparameter settings of those that are varied in an experiment
  - My naming scheme: model-hparam1_value1-hparam2_value2
  - We will see later why this is useful in TensorBoard

HSLU

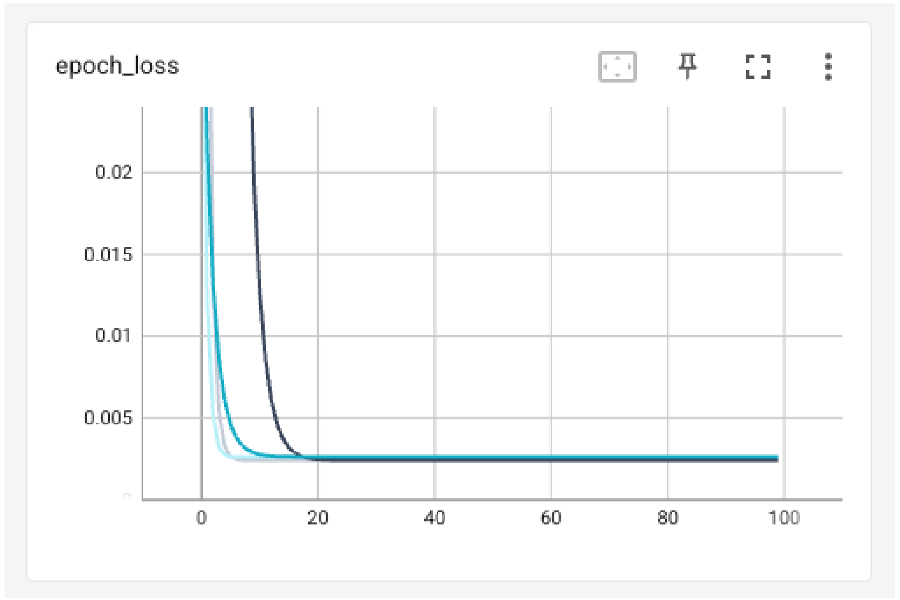
Folder structure



```
amarfurt@ssh01:~$ ls -l temp/logs/curation_corpus/slot-attention/slot/l0-drop/
total 64
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.001-l0_drop_warmup_0.1
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.001-l0_drop_warmup_0.2
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.001-l0_drop_warmup_0.5
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.01-l0_drop_warmup_0.1
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.01-l0_drop_warmup_0.2
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.01-l0_drop_warmup_0.2-l0_drop_dimensions
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.01-l0_drop_warmup_0.5
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.01-l0_drop_warmup_0.5-l0_drop_dimensions
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.1-l0_drop_warmup_0.1
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.1-l0_drop_warmup_0.2
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.1-l0_drop_warmup_0.2-l0_drop_dimensions
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.1-l0_drop_warmup_0.5
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_0.1-l0_drop_warmup_0.5-l0_drop_dimensions
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_1-l0_drop_warmup_0.1
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_1-l0_drop_warmup_0.2
drwxr-xr-x 3 amarfurt idiap 4096 Jan 12  2022 slot-use_l0_drop-l0_drop_lambda_1-l0_drop_warmup_0.5
```

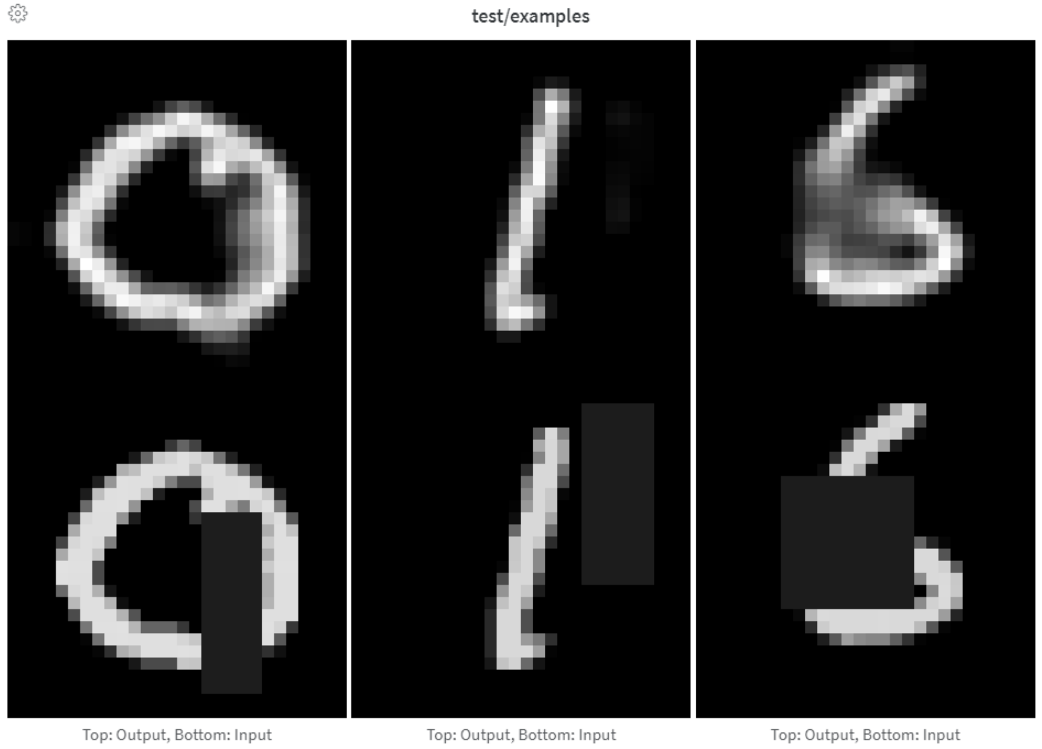
Scalar metrics

- Everything expressed as single numbers: loss, accuracy, F1, ROUGE, ...
- This is most likely your go-to for deciding which model to select
  - Think carefully about them
- Can be used for early stopping



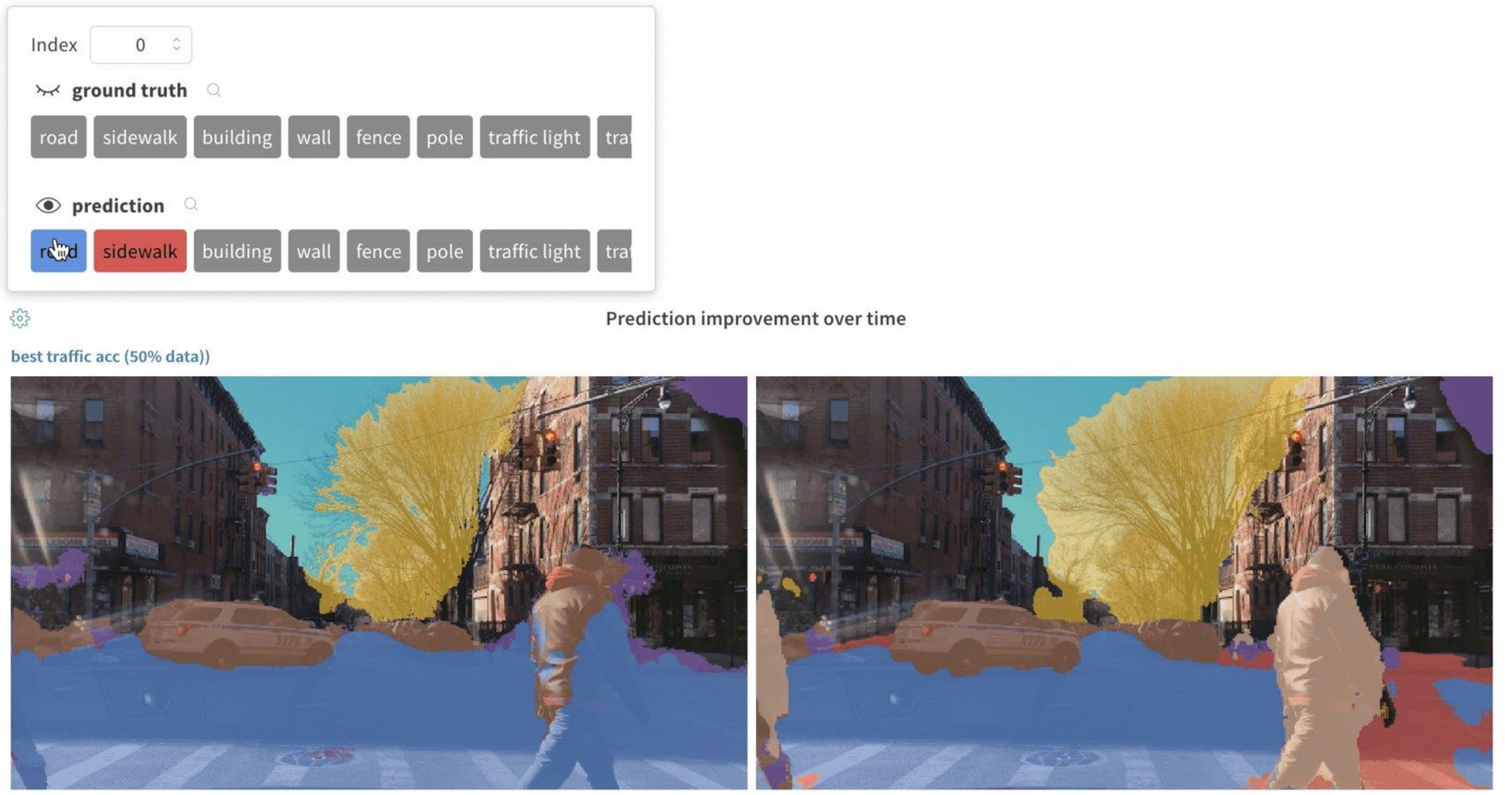
Text/image outputs

- Can also add model-generated text or image outputs to see how the outputs change over training epochs


test/examples
Top: Output, Bottom: Input
Top: Output, Bottom: Input
Top: Output, Bottom: Input

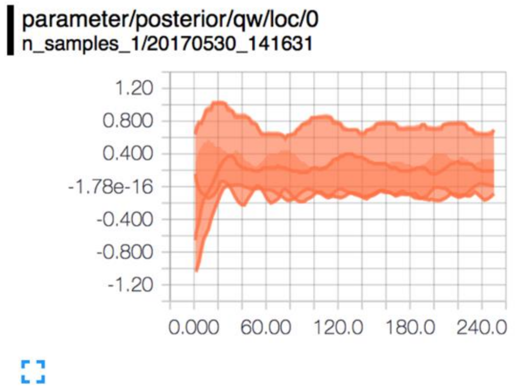
Advanced output display

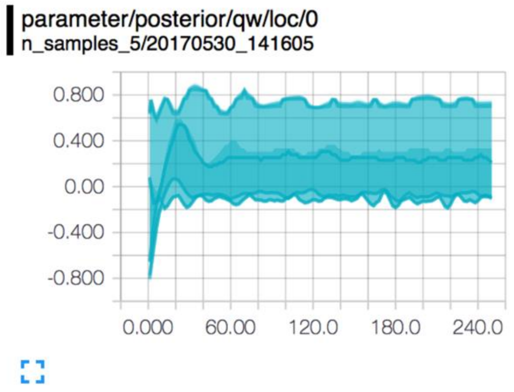
- Add interactive visualizations (e.g. segmentation masks)



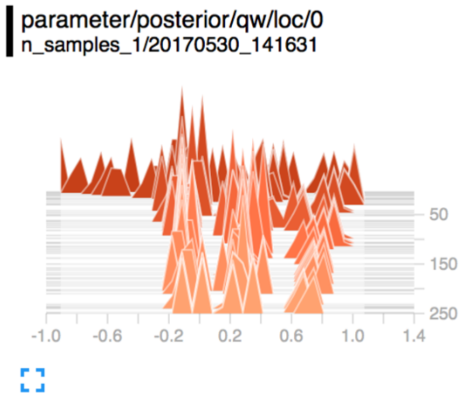
Distributions/histograms

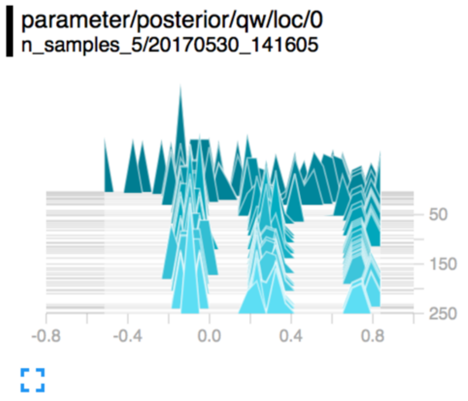
- How the model's parameters evolve over training
- Distributions:





- Histograms:

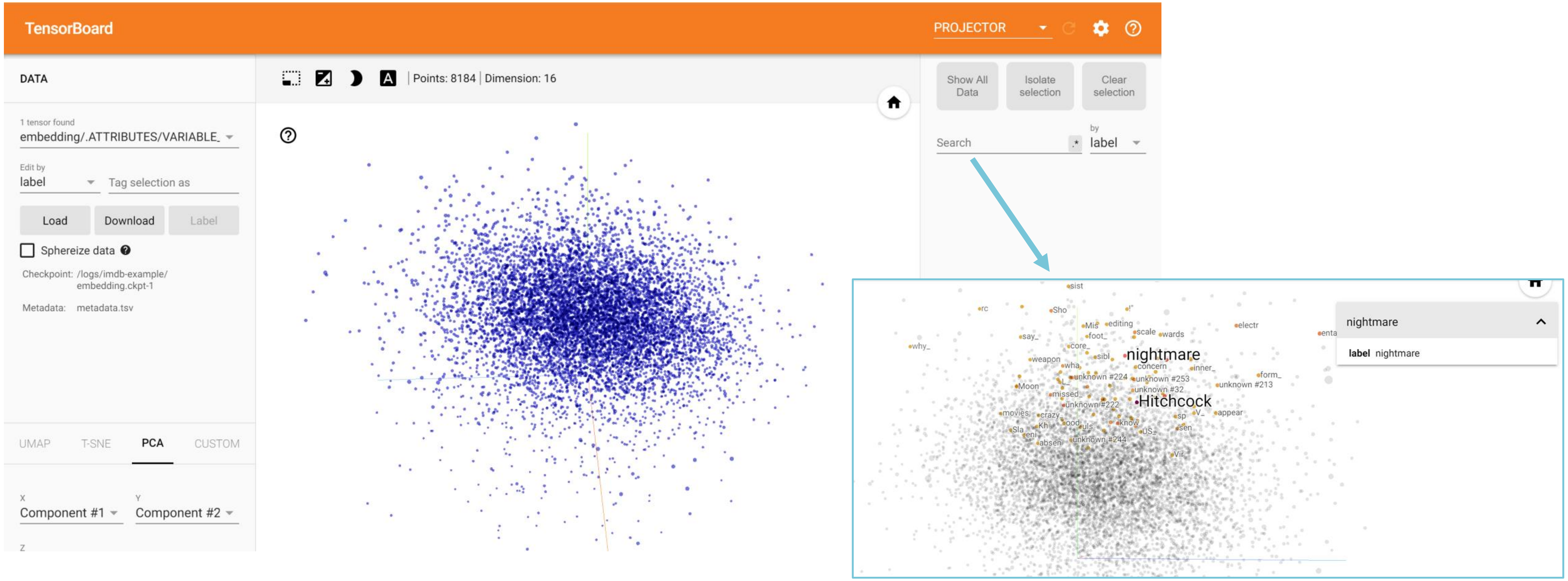




Are they in the range we expect them to be?

Embedding visualizations

- Can visualize embeddings with 2D projections (PCA, t-SNE, …)



1 tensor found
embedding/.ATTRIBUTES/VARIABLE_

Edit by
label

Checkpoint: /logs/imdb-example/embedding.ckpt-1

Metadata: metadata.tsv

Points: 8184 | Dimension: 16

X Component #1
Y Component #2

by label

label nightmare

HSLU https://www.tensorflow.org/tensorboard/tensorboard_projector_plugin

TensorBoard Demo

- Demo
  - Start TensorBoard
  - Open http://localhost:6006/#scalars

- Play around with it yourself in a Jupyter notebook: https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/scalars_and_keras.ipynb

Weights and Biases

- https://wandb.ai/stacey/deep-drive/workspace?workspace=user-lavanyashukla
- https://wandb.ai/epflmlo-epfl/swissai-eval-main-v1.5?nw=nwuserepflmlo

- W&B's own courses
- DeepLearning.AI course on evaluating and debugging generative AI

Versioning

- Need to keep track of which version of data, model and code was used to run an experiment
- Full lecture on this later

Hyperparameter optimization

Hyperparameter optimization

- Parameters vs. hyperparameters
- Unfortunately still very important to get the best out of your model
- Small differences in hyperparameters can have a big impact on model performance
- Different ways to look at this problem
  - What is the best performance I can reach for my model & dataset? (usual)
  - How well does my model perform on average under different hyperparameter settings (how robust is it)?
  - What is the best performance I can reach given some training budget?

→ Depending on what you analyze, you report best/average performance of N experiments

Hyperparameter tuning methods

Manual tuning:

- Start with a best guess at reasonable hyperparameter values (look at previous work with similar models/data)
- Observe result
- Adjust hyperparameters
- Repeat

Wait for training to finish for each result → slow

Grid search

Most basic hyperparameter optimization

1. Select a set of hyperparameters to optimize
2. Select a set of values for each hyperparameter to test
3. Build cartesian product and run each trial

Example:
Learning rate (1e-3, 1e-4, 1e-5), dropout probability (0.1, 0.2), layer norm (yes/no) → 3x2x2 = 12 total experiments

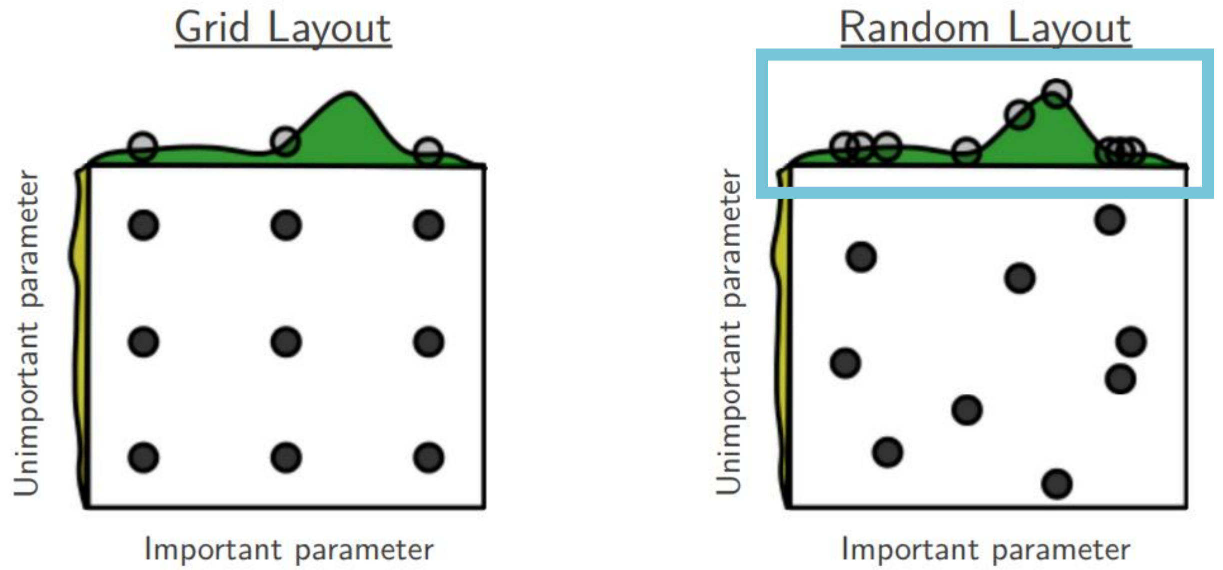
Experiment 1: lr 1e-3, dropout 0.1, no layer norm

Experiment 2: lr 1e-3, dropout 0.1, with layer norm

…

Random search

- Bergstra and Bengio, 2012
- Random search can cover more values of the important parameter:

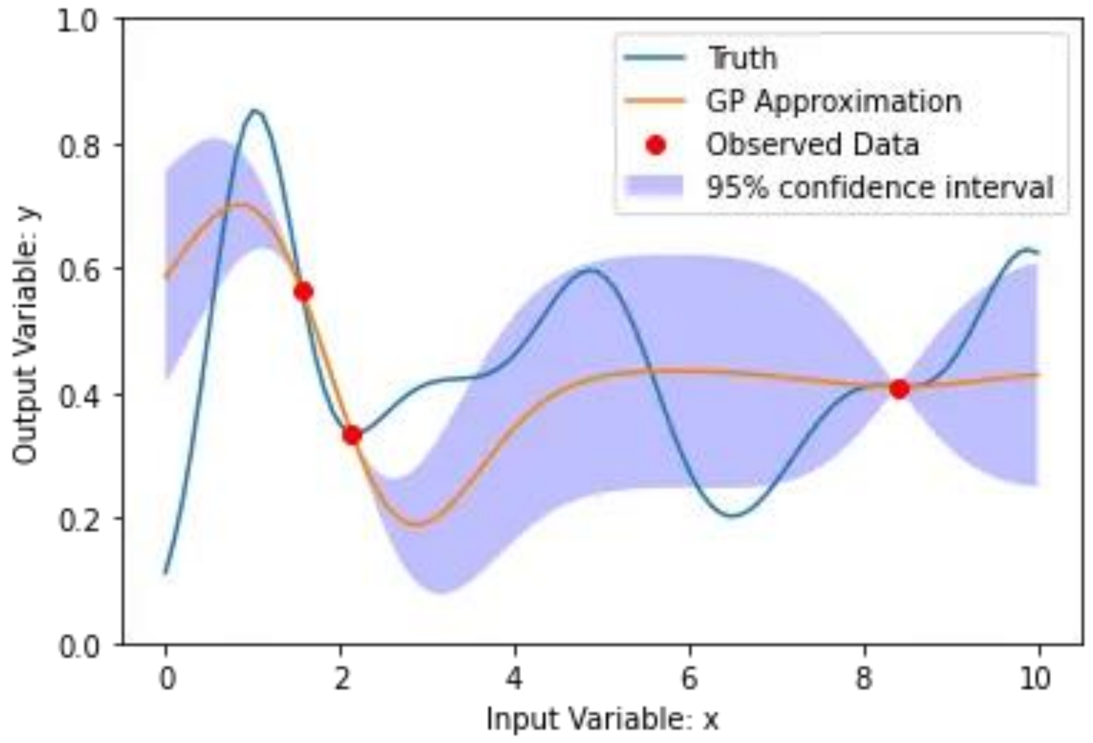


Bayesian optimization

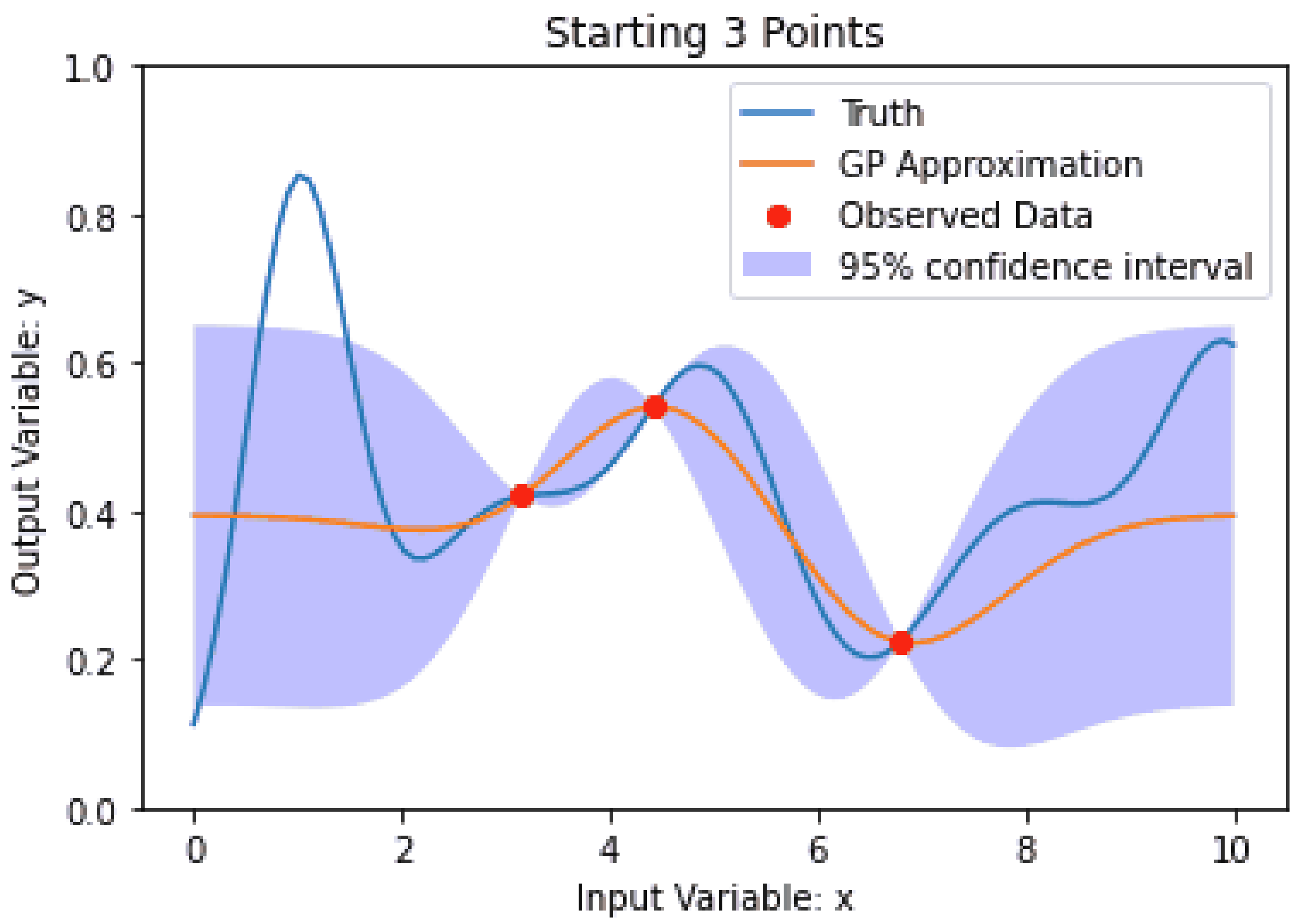
- Grid/random search predefine the set of hyperparameter values to test
- Bayesian optimization uses past results to determine which values to experiment with next
- Different methods to select values:
  - Tree-structured Parzen Estimators (TPE)
  - Gaussian Processes
  - Random Forest Regressions

Exploration vs. exploitation

- Exploration: decrease uncertainty in underexplored areas
- Exploitation: make use of knowledge about well-performing areas

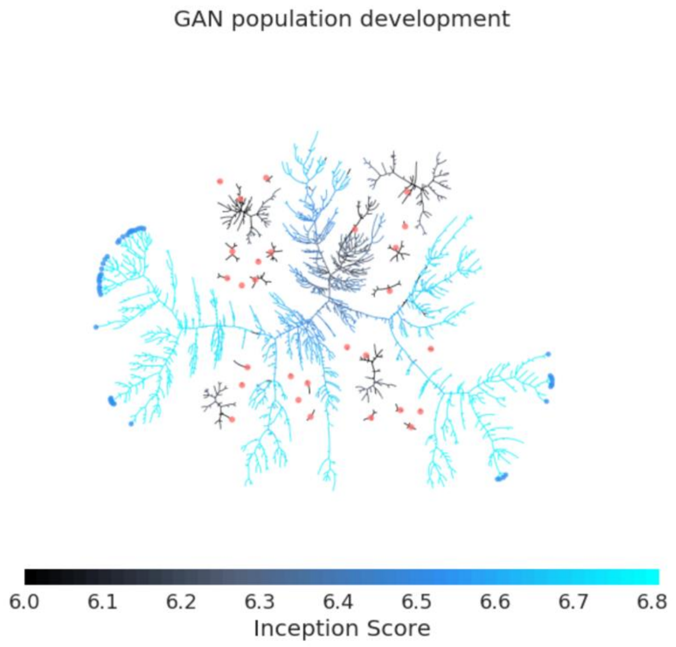


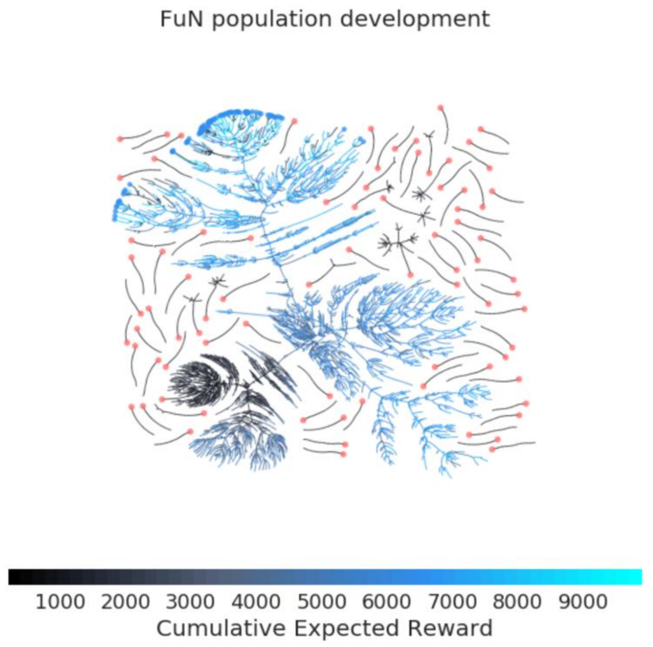
Exploration vs. exploitation



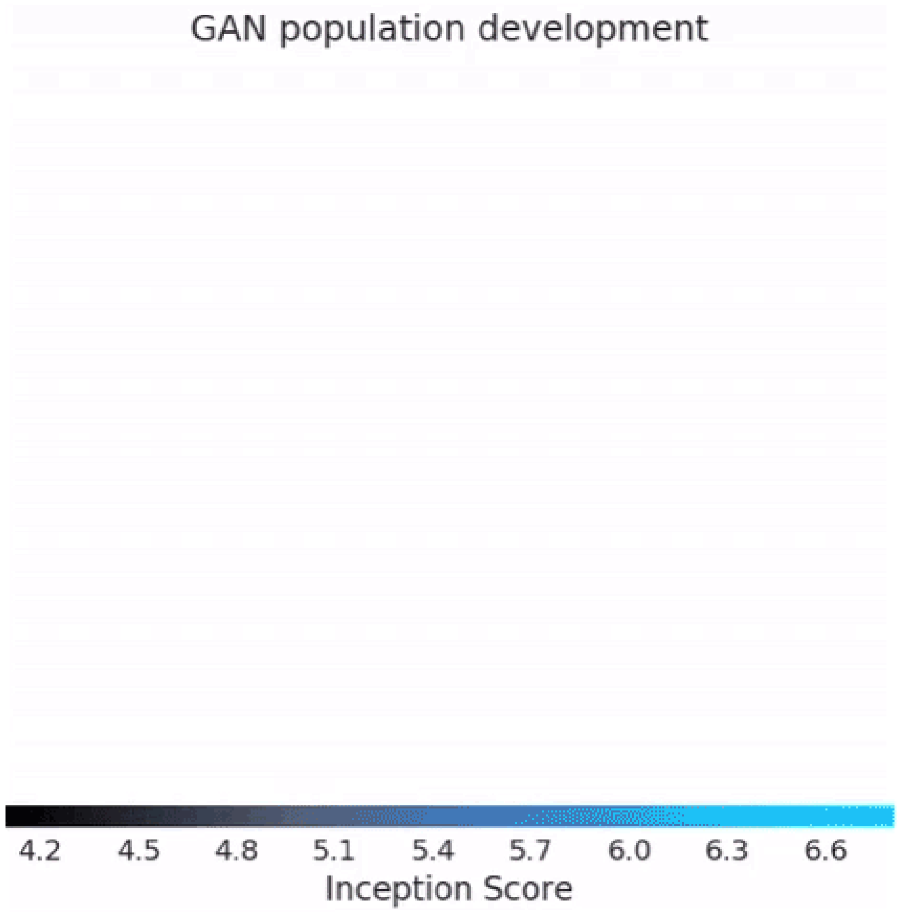
Evolutionary algorithms: Population-based training (PBT)

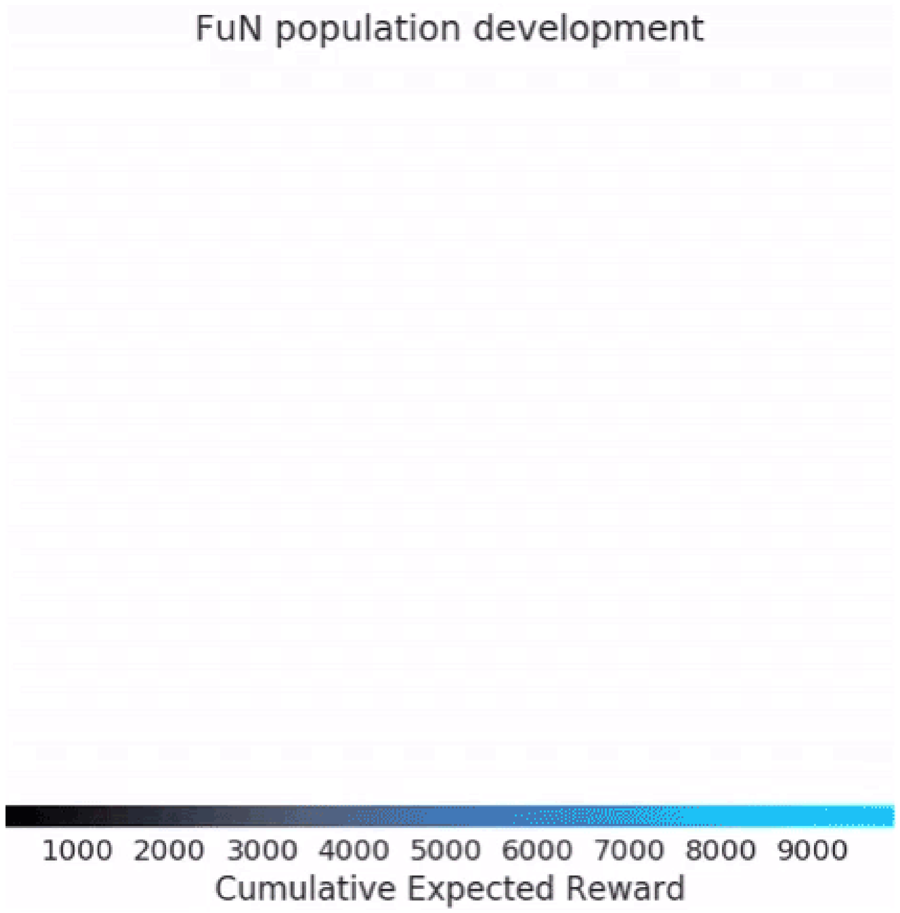
- Start with multiple hyperparameter settings as in random search
- Kill ("prune") training runs of underperforming settings (red dots)
  - Restart from well-performing checkpoints
- Explore areas of good hyperparameter values more thoroughly (blue dots)





Population-based training in action





Practical advice: batch size

- Choose the maximum batch size for your model & hardware (memory) combination, such that you don't run out of memory
- If you have different lengths in your training examples, make sure that the maximum possible length of a batch still fits in memory
    - Otherwise, you'll schedule a training run over night and get an out-of-memory error towards the end of your first training epoch, when you've already gone to bed

Practical advice: hyperparameter value ranges

Example: learning rate

- Don't test these values: [1e-5, 2e-5, 3e-5]
- Do test exponentially changing values: [1e-4, 1e-5, 1e-6]
    - Same idea: When you get more fine-grained, test 1e-4.5 and not 5e-4 ("binary search in exponential hyperparameter space")

Practical advice: hyperparameter value ranges

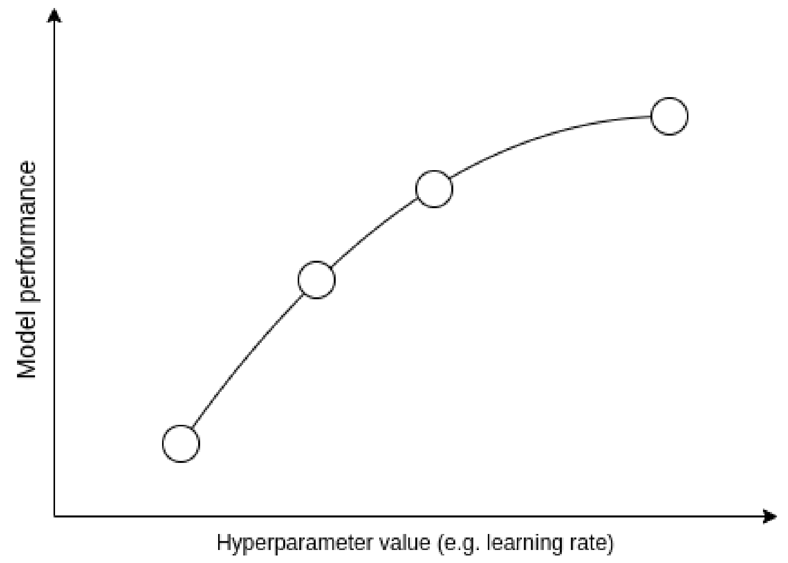
Example: learning rate

- Don't test these values: [1e-5, 2e-5, 3e-5]
- Do test exponentially changing values: [1e-4, 1e-5, 1e-6]
  - Same idea: When you get more fine-grained, test 1e-4.5 and not 5e-4 ("binary search in exponential hyperparameter space")

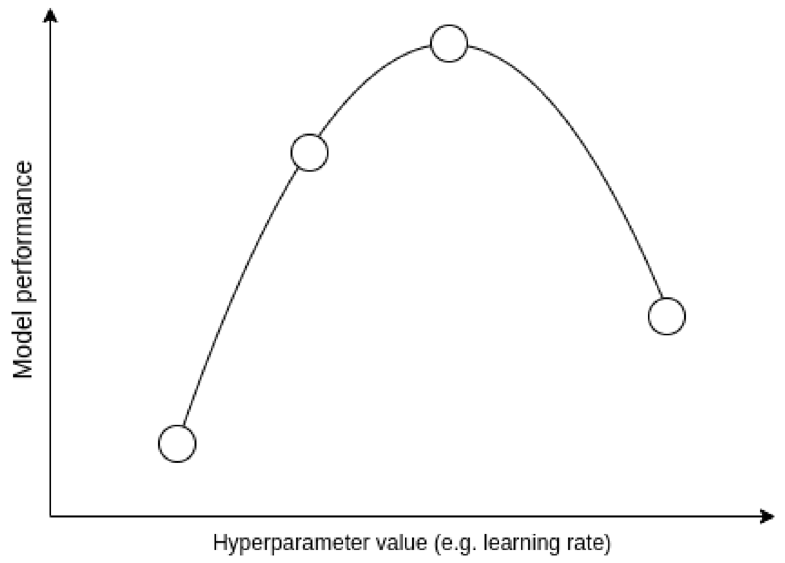
Tell your friends this phrase now makes sense to you. ;-)

- Don't stop testing hyperparameters values when the best performance has a value on the boundary of your testing range
  - I.e. learning rate of 1e-6 gives the best performance in the above example
  - Reason: Maybe 1e-7 would give even better results?

Practical advice: hyperparameter value ranges



bad

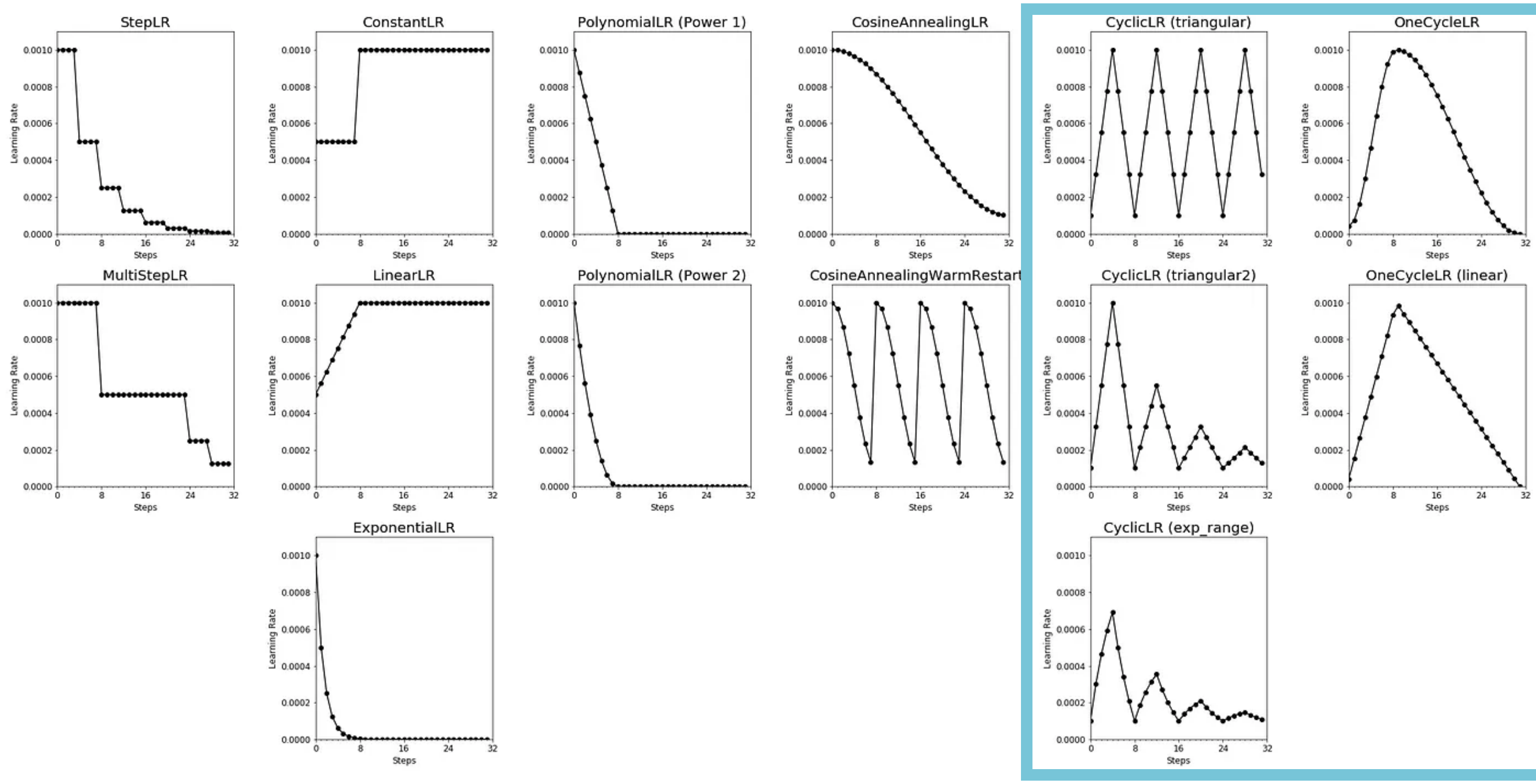


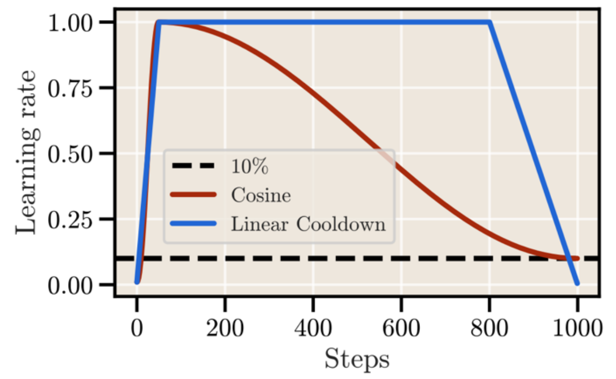
good

HSLU

Practical advice: learning rate schedule

- For Transformers, I've had most success with schedules with a cyclical schedule (includes a warmup and a subsequent decay), usually 1 cycle





New research shows you can also keep the learning rate constant

HSLU A Visual Guide to Learning Rate Schedulers in PyTorch; Hägele et al., 2024 33

Practical advice: How many trials should you run?

- Depends on your training budget: Most benefits?
  - Different types of models
  - Different hyperparameter settings
  - Training for longer/on more data (data augmentation)
  - Different finetuning techniques
  - Ensembling

$\rightarrow$ Empirical question

Practical advice: Should you tune the random seed?

- Debated question
- Pro
  - It's just another hyperparameter
  - It influences "valid" hyperparameters, such as initialization and the order of training examples
- Contra
  - Performance shouldn't rely on such meaningless hyperparameters → shows the model is not robust
  - Best random seed depends on library versions/hardware (→ are these hyperparameters, too?)

Hyperparameter optimization tools

- Hyperparameter sweeps in W&B
  - Start with defining a sweep configuration
- Hyperparameter visualization in TensorBoard
- Dedicated libraries for automated hyperparameter tuning
  - Optuna
  - Hyperopt
  - Ray Tune
  - Vizier
  - many more...

Automating hyperparameter tuning

- In-depth article comparing Optuna and Hyperopt
    - Spoiler: Optuna comes out on top in all categories
- Optuna [Website] [Tutorials]
    - Works with all frameworks and several training libraries (e.g. PyTorch Lightning)
    - Quick integration into code:

```python
import optuna
def objective(trial):
    x = trial.suggest_float('x', -10, 10)
    return (x - 2) ** 2
study = optuna.create_study()
study.optimize(objective, n_trials=100)
study.best_params  # E.g. {'x': 2.002108042}
```

Aside: Neural architecture search (NAS)

- The neural network architecture is just another set of hyperparameters that can be optimized
- Define a set of architecture components as the search space
- Let an algorithm select the best combination by defining an architecture, training the model, and observing the final loss

Further reading:

- Survey with insights from looking at 1000 NAS papers
- Pioneering paper on neural architecture search with reinforcement learning (2016)
- Recent combinatorial optimization NAS paper from Google

HSLU

Hyperparameter optimization links

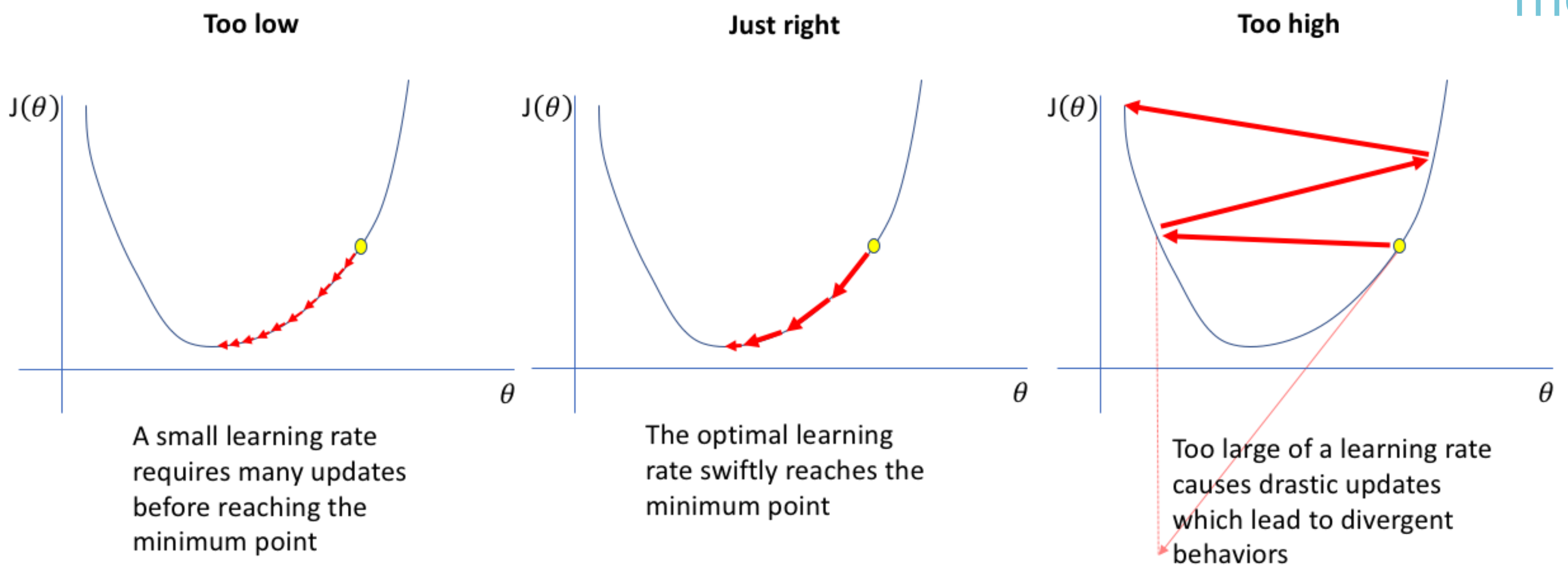
- Deep dive into hyperparameter optimization by Amazon
- Deep Learning Tuning Playbook by Google
- Neptune.ai blog post with a list of 12 hyperparameter optimization libraries

Learning curves

Learning in a neural network

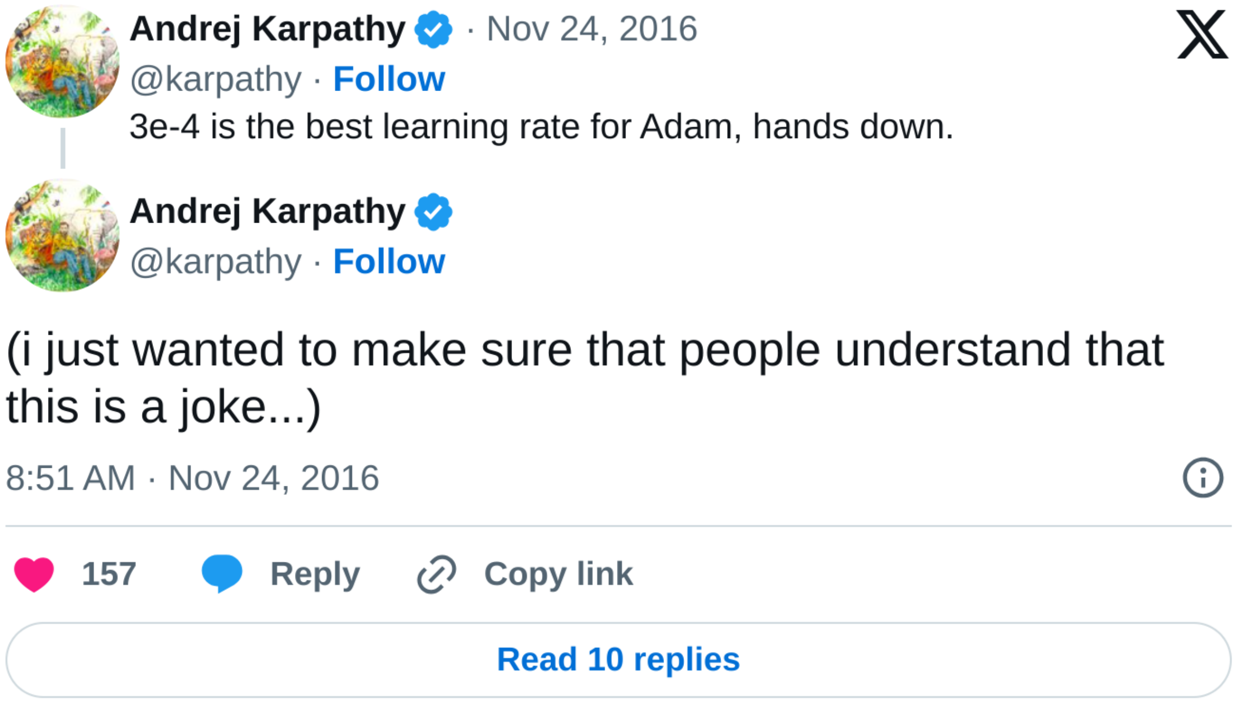
- Goal of optimization: find (a local) minimum of objective function
- Optimizers have hyperparameters, too: learning rate, momentum, weight decay

most important


Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Learning in a neural network

- It's not that simple...



Andrej Karpathy · Nov 24, 2016
@karpathy · Follow
3e-4 is the best learning rate for Adam, hands down.

Andrej Karpathy
@karpathy · Follow

(i just wanted to make sure that people understand that this is a joke...)

8:51 AM · Nov 24, 2016

157 Reply Copy link

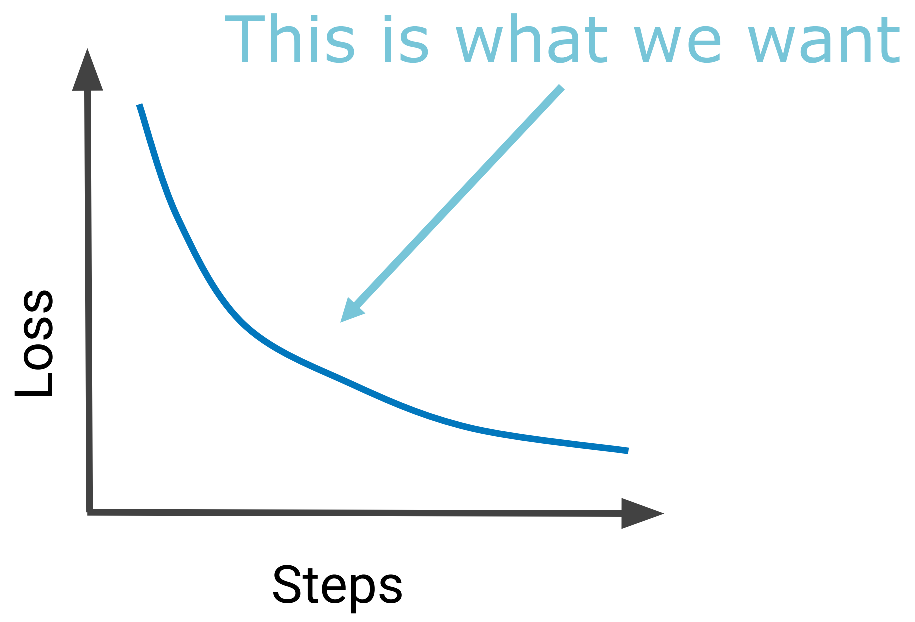
Read 10 replies

Learning/loss curves

Looking at the loss is usually the first step of debugging.

Common problems:

- Loss does not decrease
- Loss oscillates/diverges
- Loss decreases too slowly
- Loss goes to 0
- Loss is negative



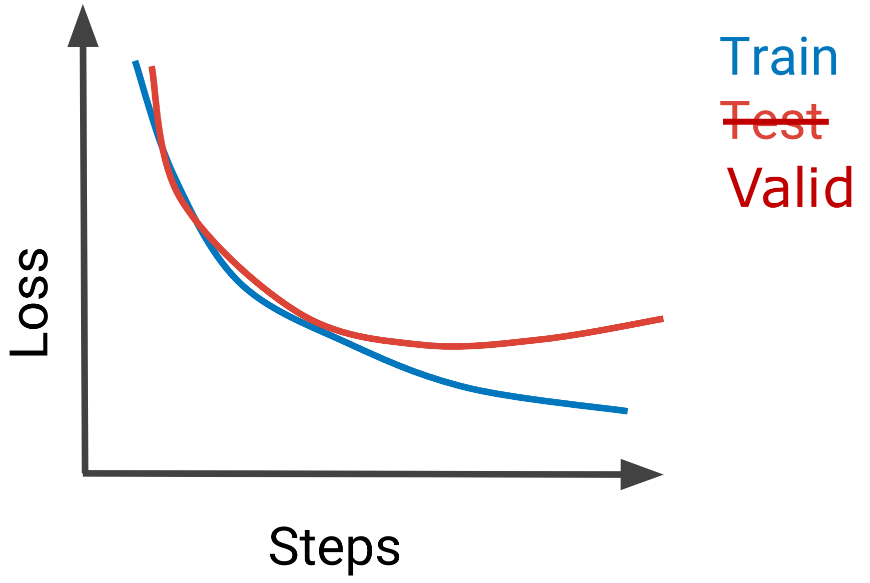
Simple problems

- Loss does not decrease
  - Loss computed incorrectly
  - Model parameters not updated (e.g. missing a loss.backward() or an optimizer.step())
  - Learning rate way too small (parameters not updated enough)
- Loss decreases too slowly: Increase learning rate
- Loss goes to 0
  - Problem deterministic/too easy
  - Model can see labels
- Loss is negative: Loss computed incorrectly

Train and validation loss diverge

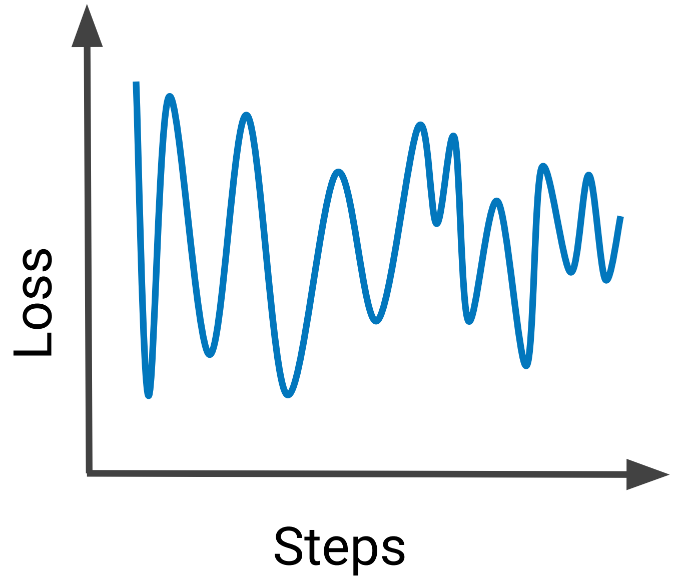
Training loss still decreases, but val loss increases = Overfitting

- Regularization
  - Reduce model capacity
  - Add explicit regularization: dropout, weight decay, batch/layer norm
  - Increase learning rate (can be tricky)
- Early stopping: Pick the model checkpoint with lowest val loss
- Do train and validation distributions match?



Loss oscillates/diverges

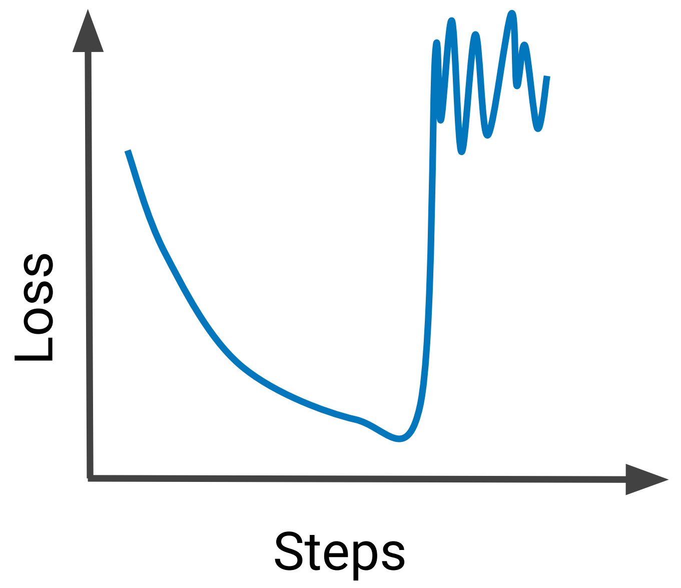
- Learning rate too high
  - Model bounces around in parameter space
- Uninformative features
  - Debugging: Can you overfit on 10 examples?
- Malformed data (wrong format?)
- Debugging:
  1) Start with a simple baseline and the same data
  2) Incrementally add complexity



Loss suddenly explodes

Exploding gradient! Causes:

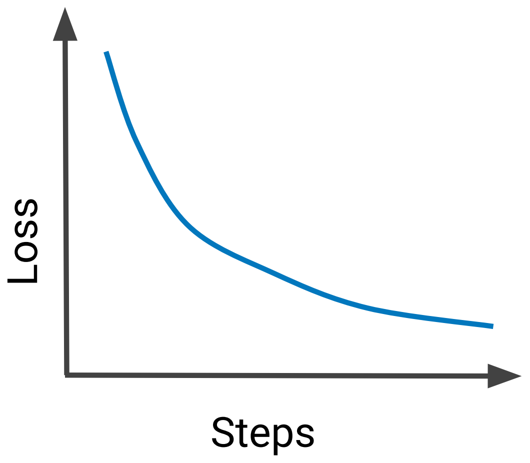
- If you use a learning rate warmup and max_lr is too high
- Data anomalies
  - NaNs
  - Very out-of-distribution data (data conversion error?)
  - Division by zero
  - Logarithm of zero or negative numbers
  - (although these will often crash your code)
- Debugging
  - Check gradient norm
  - Gradient clipping ("treating the symptom, not the disease") → better check if data is well-formed

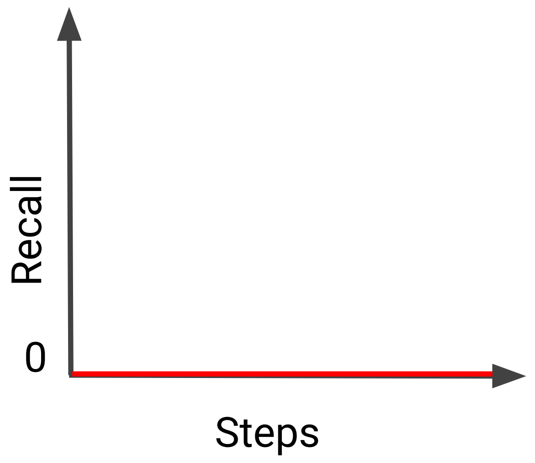


Disagreeing metrics

0 Recall = no example classified as positive

- Classification threshold too high? → Check distribution and adapt threshold
- Class imbalance in training data? → Data augmentation with more positive examples
- Check threshold-invariant metrics, such as area under curve (AUC)

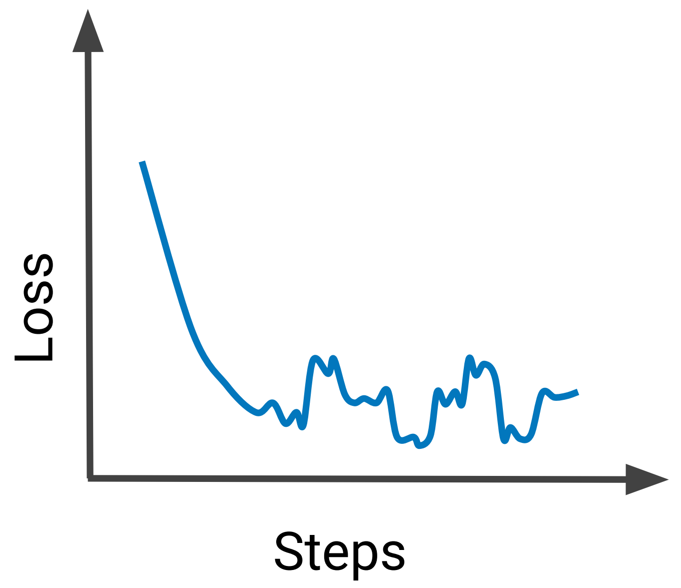




Training loss no longer decreases

Loss shows repetitive, step-like behavior

- Some examples are harder = higher loss
  - Model no longer updated b/c learning rate too small?
- Constant learning rate: Model cannot generalize and goes through the same updates again and again?
  - Use a learning rate schedule
- Data ordering effects
  - Shuffle your data (potentially after every epoch, but not always)

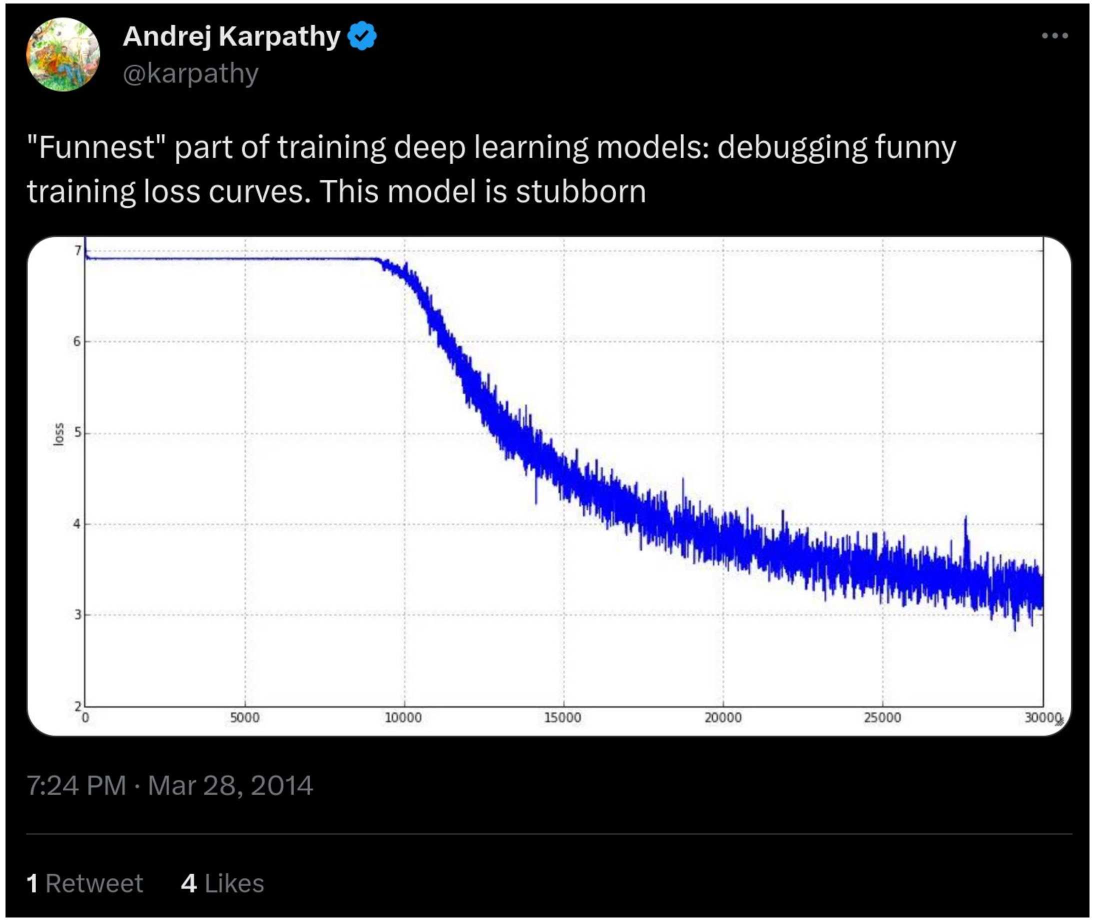


If the loss curve is unclear...

Look at your model's outputs!!!

(can't be stressed enough)

Loss plateaus, emerging abilities



Andrej Karpathy
@karpathy

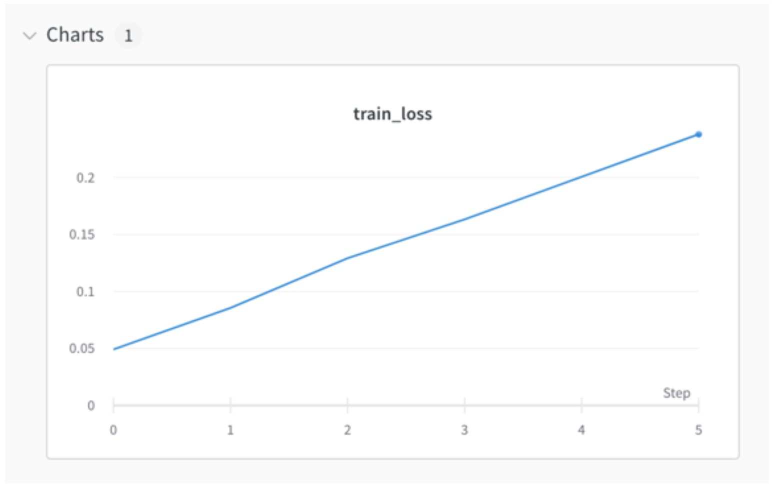
"Funnest" part of training deep learning models: debugging funny training loss curves. This model is stubborn

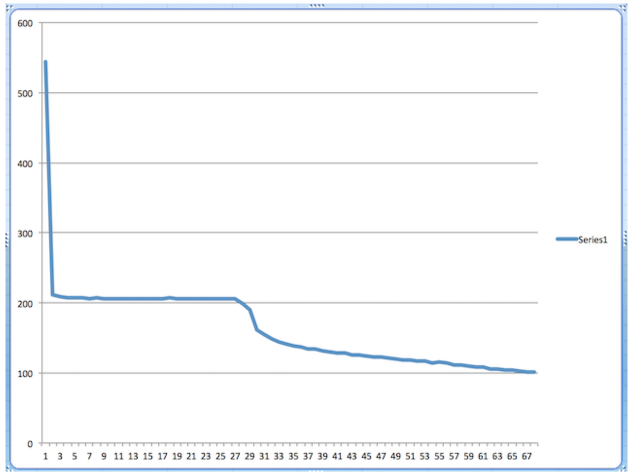
7:24 PM · Mar 28, 2014

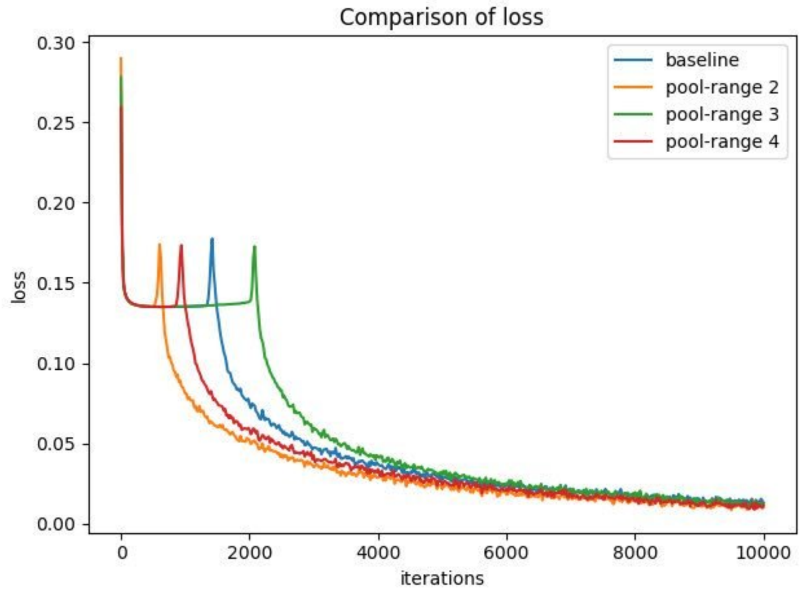
1 Retweet 4 Likes

Interesting loss curves

- Loss functions Tumblr:







- Public training runs, such as GPT-NeoX 20B pretraining (WandB)

HSLU

Good resources

- Crash course: Google Developers - Interpreting loss curves
- Advanced: Deep Learning Tuning Playbook by Google
- LLM Training (best resource on the topic that we have): https://github.com/facebookresearch/metaseq/tree/main/projects/OPT/chronicles
  - (includes the full notes on OPT-175B training, a GPT-3 175B reproduction)
- A lot of info on LLM training of the Swiss LLM Apertus