

Project Discussions

NLP
Andreas Marfurt

Schedule

- Week 1: Experiment tracking
- Week 2: Preprocessing
- Week 3: Data loading, batching
- Week 4: Input/output format
- Week 5: Training (checkpointing, early stopping)
- Week 6: Hyperparameter tuning
- Week 7: Evaluation (metrics, error analysis)
- Week 10: Debugging, testing
- Week 11: Interpretation & expectations
- Week 12: Jupyter notebooks
- Week 13: Presentations

Week 1: Experiment Tracking

Experiment Tracking

- You will run a lot of experiments to find the best hyperparameters
- Use a tool to keep track of them and compare
- Most popular choice: Weights and Biases
- Offline option: TensorBoard
 - Developed with TensorFlow, standalone project now
 - Can be used with PyTorch, too



We also use W&B

Demo: TensorBoard

Exercise: Experiment Tracking

Week 2: Preprocessing

Preprocessing decisions

- See notebook checklist

Week 3: Data Loading

Batching

- Instead of single examples, process multiple examples together
- Improves efficiency and optimization stability
 - Outliers have a smaller impact in larger batches
- Batch size: Number of examples in a batch
- Padding: Pad all examples to the same length
 - Example: Translating multiple sentences
 - For all but the longest sequence, add <pad> to the end of the sentence
 - <pad> tokens are ignored for computation (e.g. attention)

Batch size

- Largest possible for GPU memory
- Larger is better for regularization within batch
 - Outliers have smaller influence
- Find by padding to longest element in train set, then do a single forward/backward pass
 - Use binary search for optimal size
- Gradient accumulation

Padding in RNNs

- https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pad_packed_sequence.html#torch.nn.utils.rnn.pad_packed_sequence

Week 4:

Input/Output Format

Input to Model

- Own vocab vs. pretrained
- Embedding
- Formatting of task
- Special tokens (separators)
- LLMs (prompt)

Input to Classifier

- Word embeddings
 - Average/sum/first token
 - Not recommended: Each token separately and average predictions afterwards
- RNNs
 - Final hidden state
- Transformers (not covered yet, will see in BERT lecture, can skip)

Output from Classifier

- Binary classification
 - 1 output with sigmoid
 - 2 outputs with softmax
- Multi-class classification
- Generated text
 - Parsing of output
 - <https://huggingface.co/blog/open-llm-leaderboard-mmlu>

Input/Output Format (Example solution)

- Input (BoolQ classification with word embeddings)
 - Average question/passage words → 2x 300-dim vectors
 - Concatenate: [passage, question]
 - Reason: Alphabetical ordering is nicer (p before q) 😊
 - 600-dim input vector to classifier
 - dimensions not necessary, but could help for understanding
 - visualizations are very useful here
- Output (Example for ImageNet classification)
 - Classifier output: Logits for 1000 classes
 - Softmax: Probability of each class
 - Select the one with highest probability

Week 5: Training

Loss

- Binary cross-entropy loss
 - Loss vs. logit loss
 - Output of training and validation step
- Cross-entropy loss
- What to use each loss for?
 - Contrastive loss
 - Margin loss

Optimizer

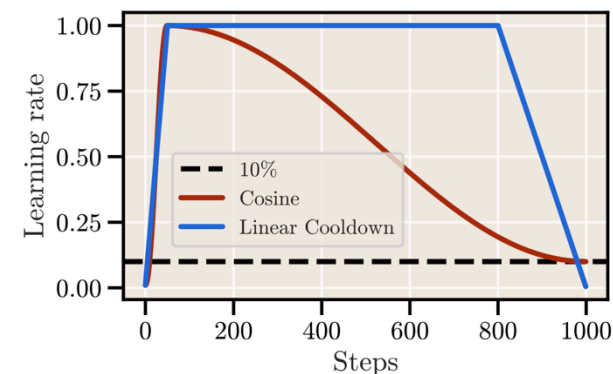
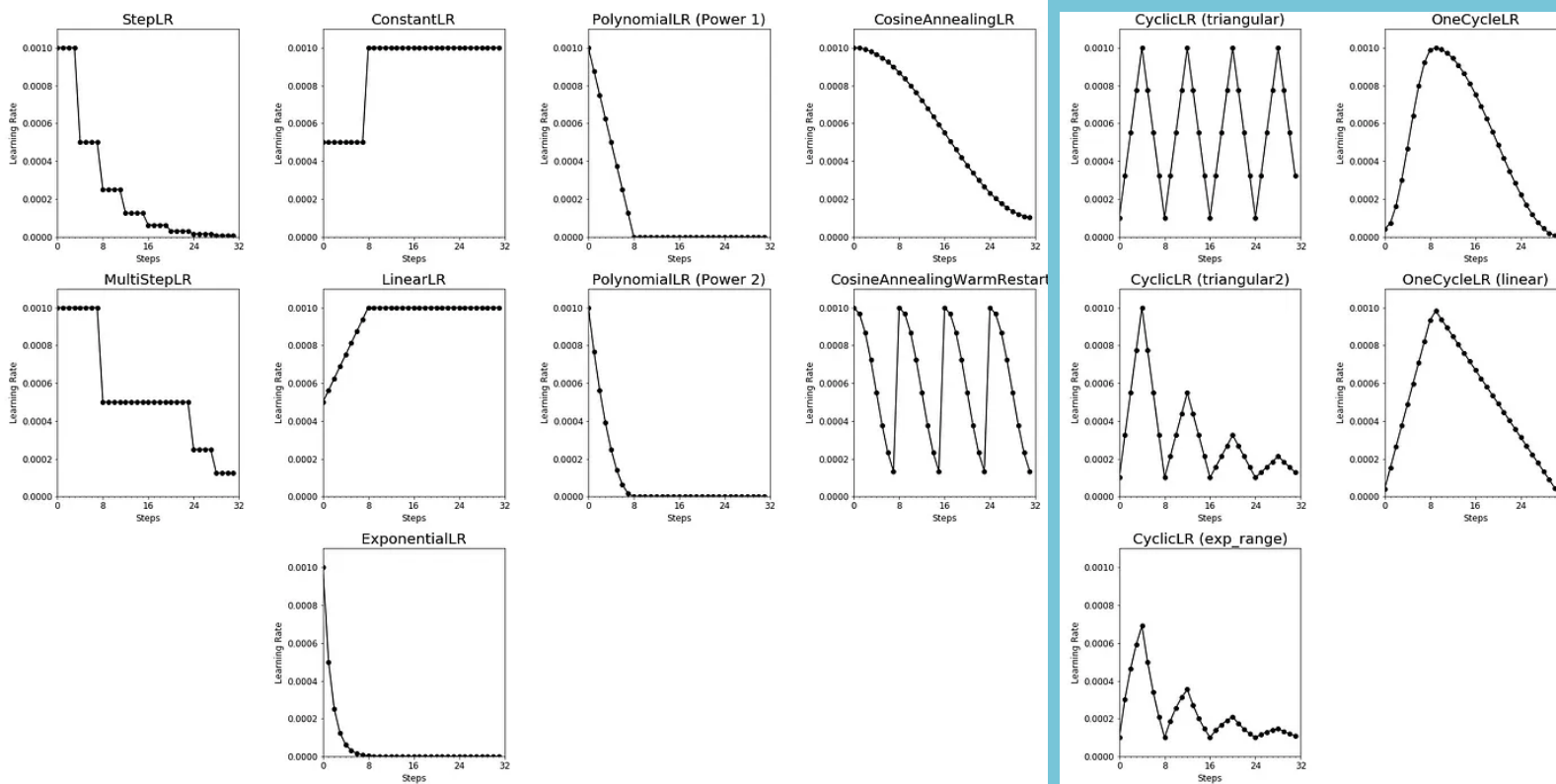
- Adam
- AdamW
- others

Learning Rate

- Initial selection
- Most important hyperparameter
- Makes sense to include (rudimentary) hyperparameter tuning in project

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

Model Checkpointing

- Why?
- How often?
 - Stable infrastructure?
 - Available storage
- Based on what metric?
- [Lightning callback](#)
- [Checkpointing guide for W&B](#)
 - Care: Storage usage

Saving model checkpoints

- Save your models, so you can run them again later
- Different methods

- Save parameters only

Save: (recommended)

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

Load:

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

vs. entire model

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

```
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](https://pytorch.org/tutorials/beginner/saving_loading_models.html))

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()
```

Early Stopping

- Why?
- Based on what metric?
- Patience
- Minimum difference (uncommon)
- Target threshold (uncommon)
- [Lightning guide](#) or [Lightning docs](#)

Week 6: Hyperparameter Optimization

Hyperparameter Optimization: 1-Slide Crash Course

- Find largest batch size that doesn't cause out of memory errors
→ Keep it fixed
- Select number of epochs that *makes sense* → Fix it
 - Difficult task: More epochs necessary to find patterns
 - A lot of training data: Can't afford to run many epochs
- Tune learning rate (usually most important hyperparameter)
 - Find area of *reasonable* learning rates by checking exponential step sizes: $1e-3$, $1e-4$, $1e-5$
- Tuning hyperparameters sequentially may miss best setting
 - But: tuning all of them together can be too many combinations
- Use early stopping: Select the model with best validation performance

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) $\rightarrow 3 \times 2 \times 2 = 12$ total experiments

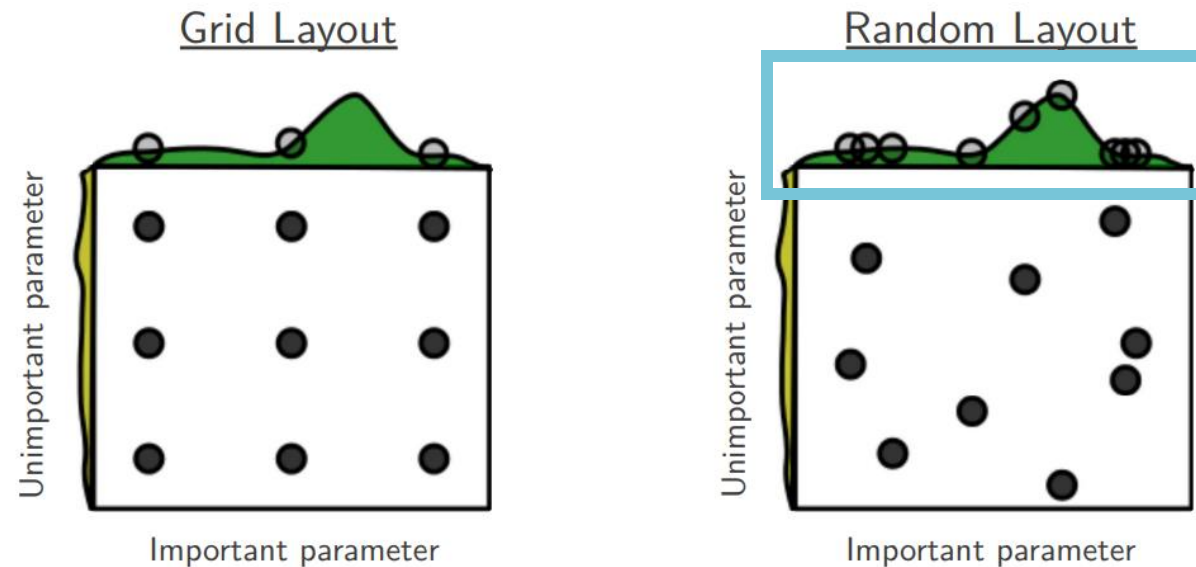
Experiment 1: lr 1e-3, dropout 0.1, no layer norm

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:

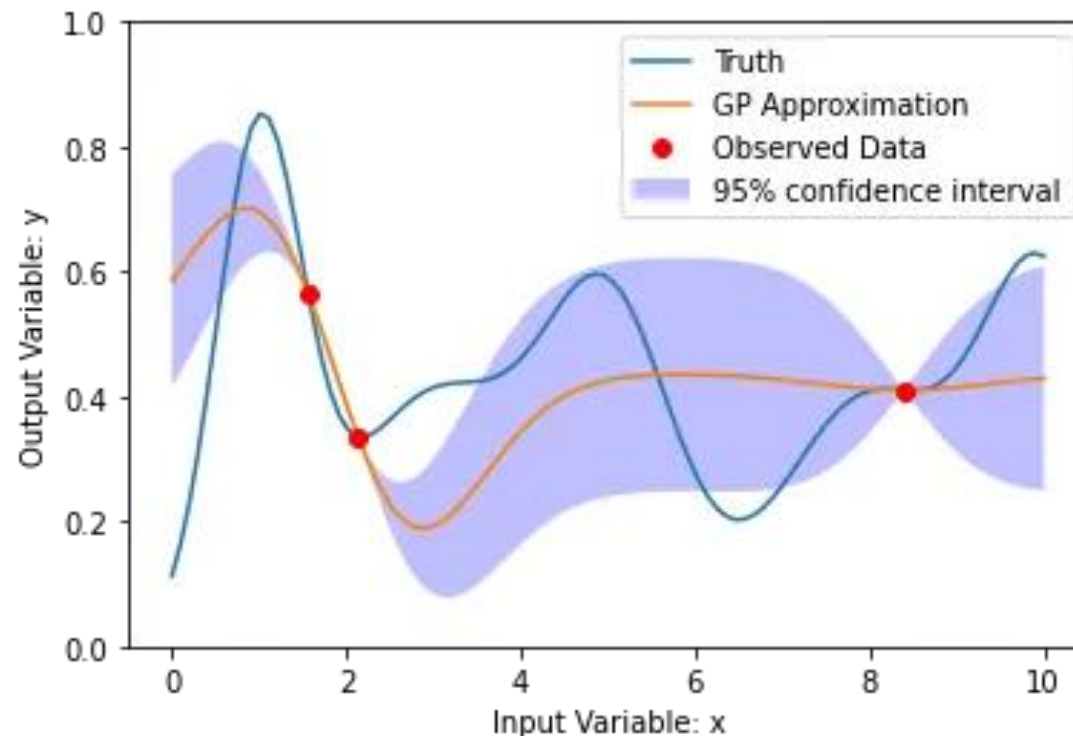


Bayesian optimization

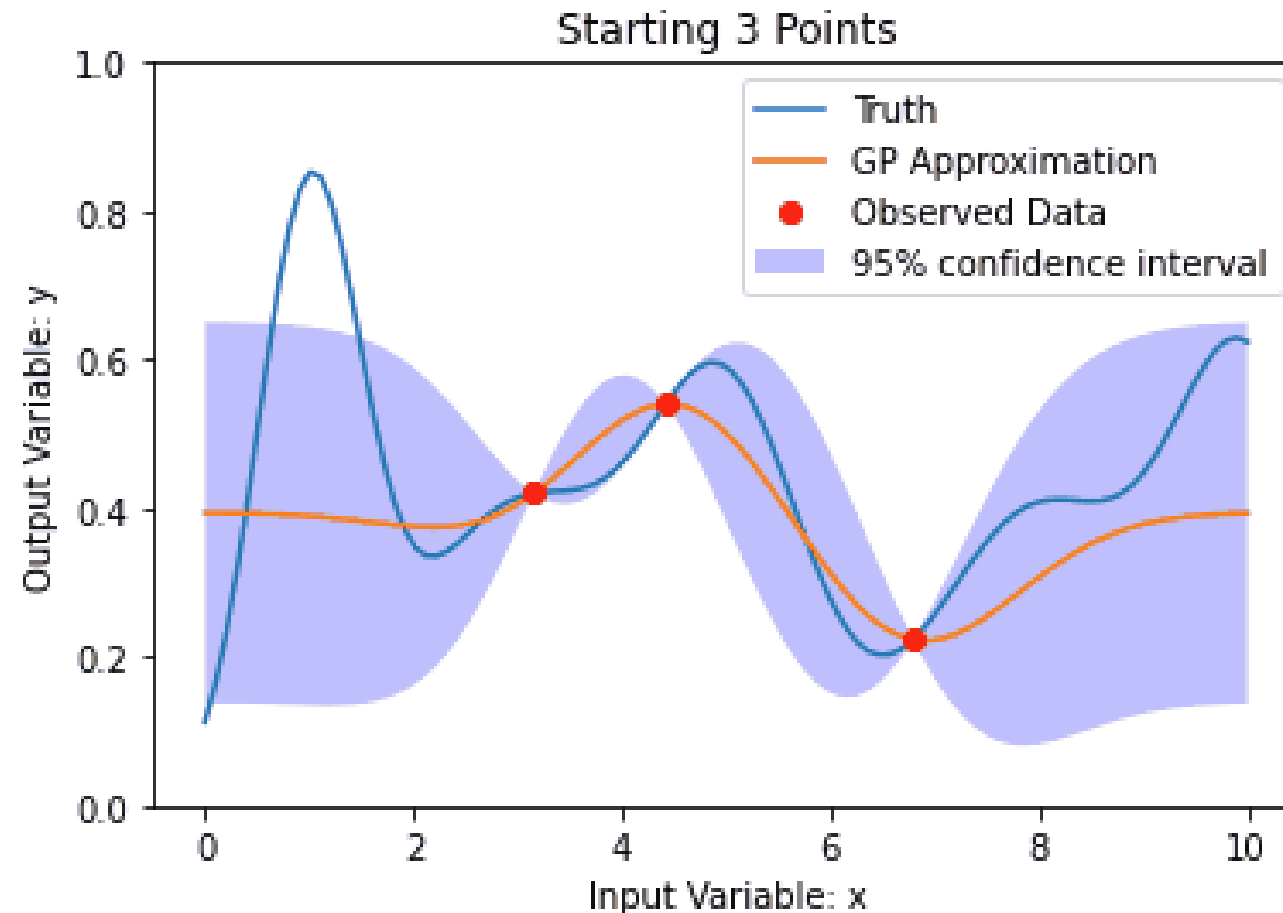
- Grid/random search predefine the set of hyperparameter values to test
- Bayesian optimization uses past results to determine which values to experiment with next
- Different methods to select values:
 - Tree-structured Parzen Estimators (TPE)
 - Gaussian Processes
 - Random Forest Regressions

Exploration vs. exploitation

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



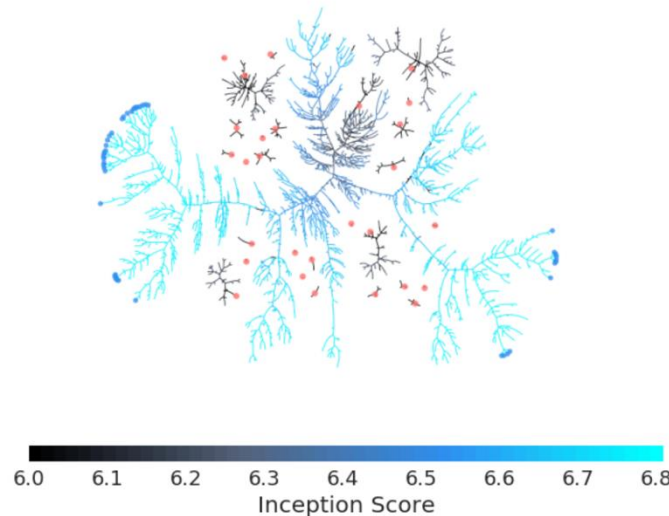
Exploration vs. exploitation



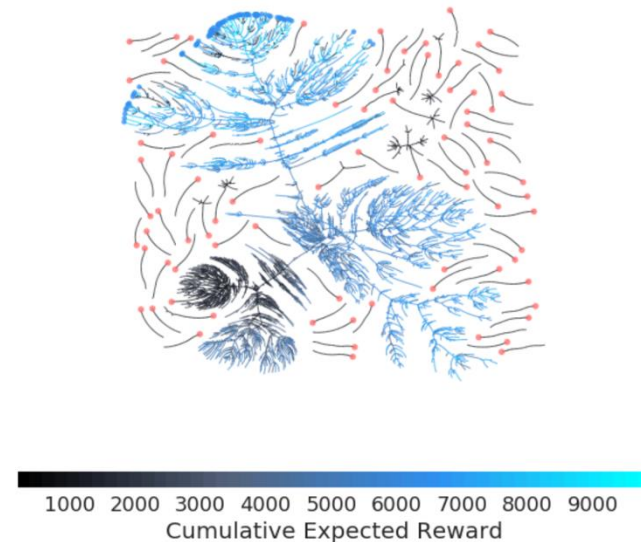
Evolutionary algorithms: Population-based training (PBT)

- 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)

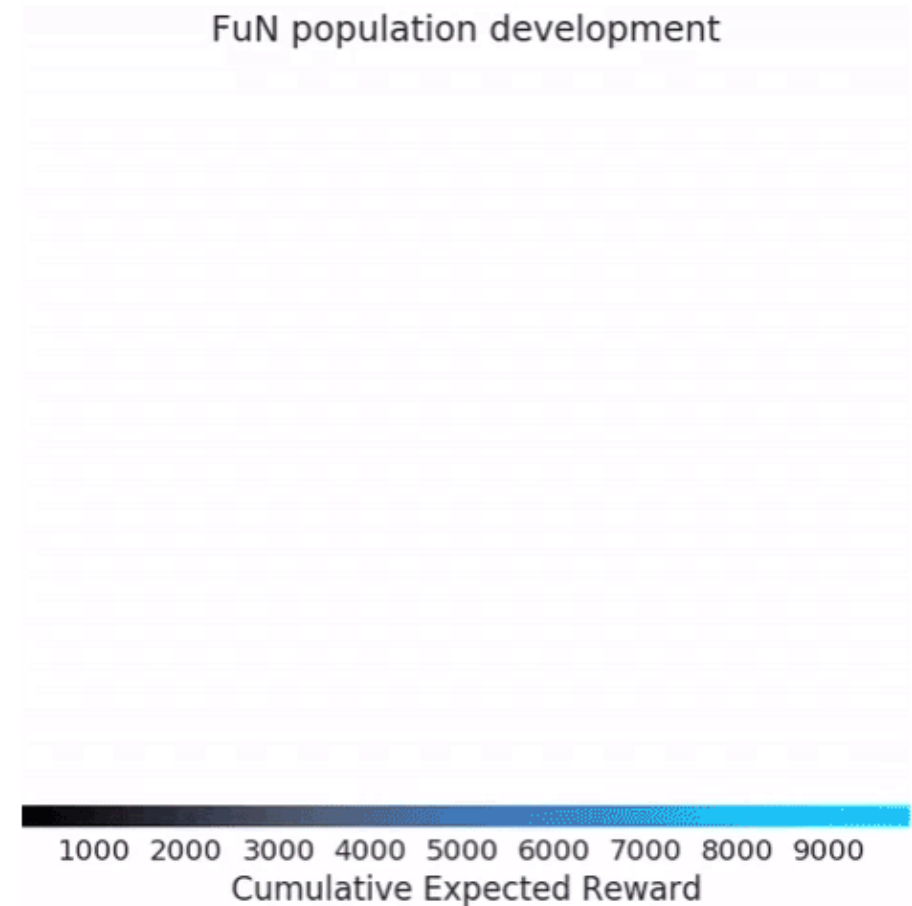
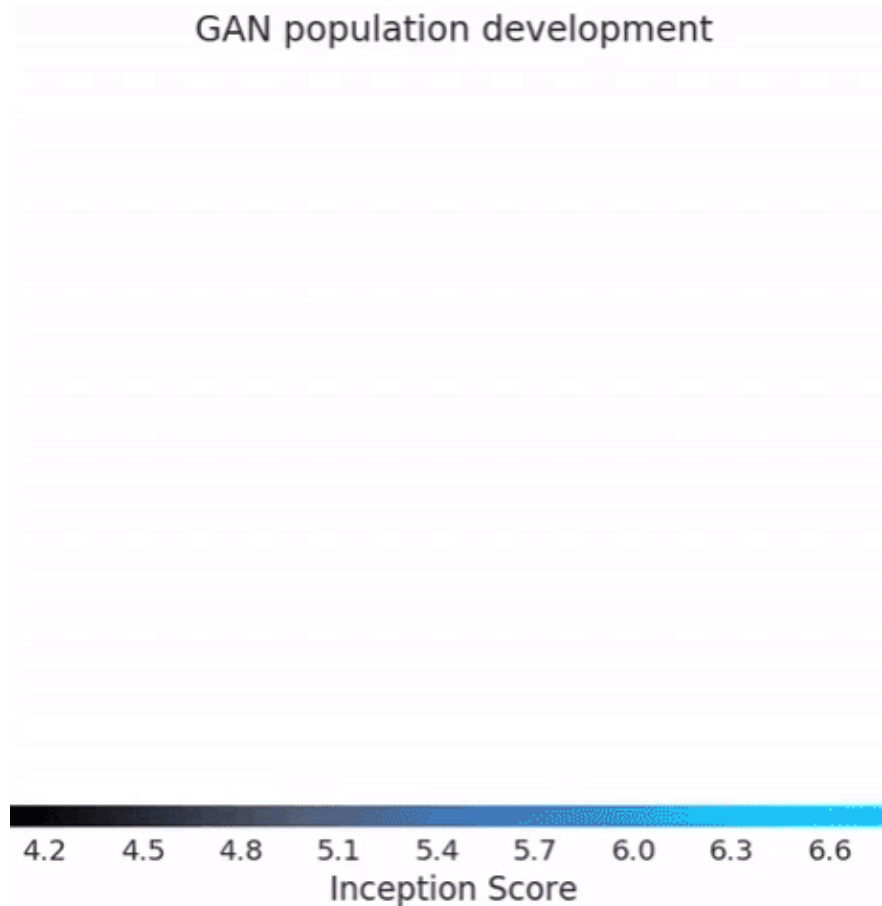
GAN population development



FuN population development



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

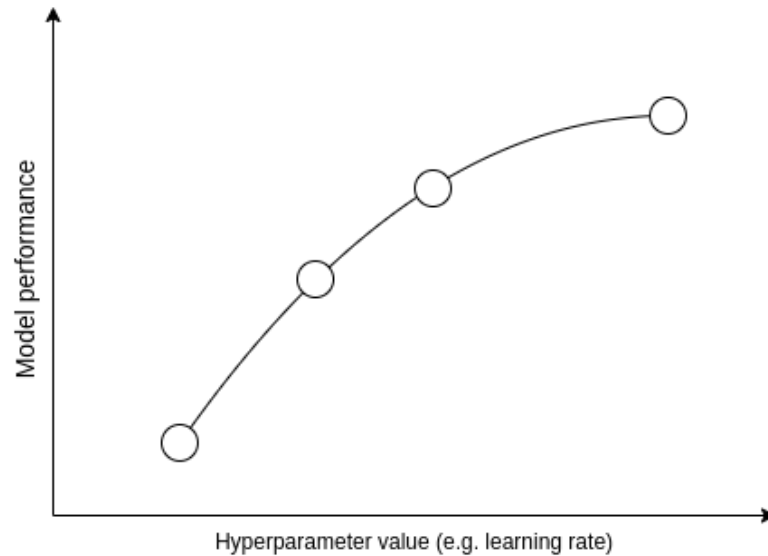
- **Don't** test these values: $[1e-5, 2e-5, 3e-5]$
- **Do** test exponentially changing values: $[1e-4, 1e-5, 1e-6]$
 - Same idea: When you get more fine-grained, test $1e-4.5$ and not $5e-4$ ("binary search in exponential hyperparameter space")

Practical advice: hyperparameter value ranges

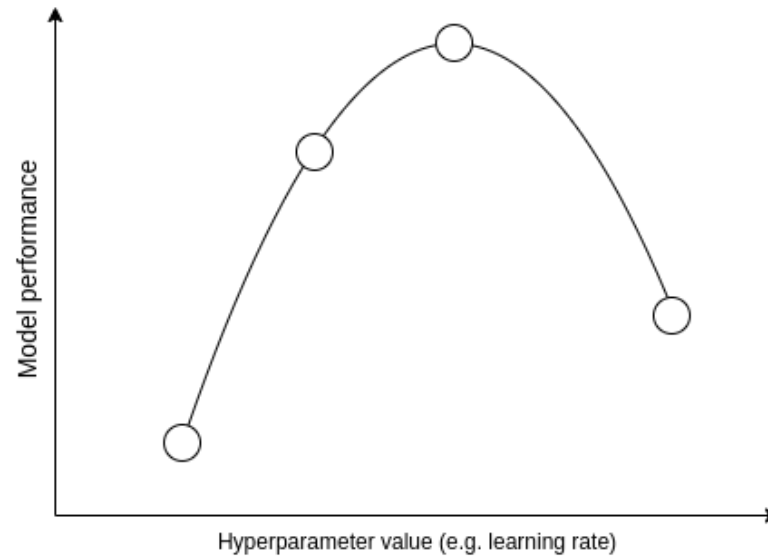
Example: learning rate

- **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

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

→ 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:

```
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](#)

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](#)

Week 7: Evaluation

Metrics (Classification)

- Accuracy
- Precision/Recall/F1
 - ~~Imbalanced dataset~~ → detection tasks
- Detection problems
 - Precision-recall curve
 - ROC
 - Area under curve
- Overfitting
 - Definition
 - Validation loss vs. accuracy

Other metrics

- Regression
 - Mean absolute/squared error, ...
- Generation tasks
 - Perplexity
 - Task-specific metrics
 - BLEU for translation
 - ROUGE for summarization
 - Token-level F1 for question answering
 - Model-based metrics
 - BERTScore, BARTScore, MAUVE, ...
 - LLM as a judge

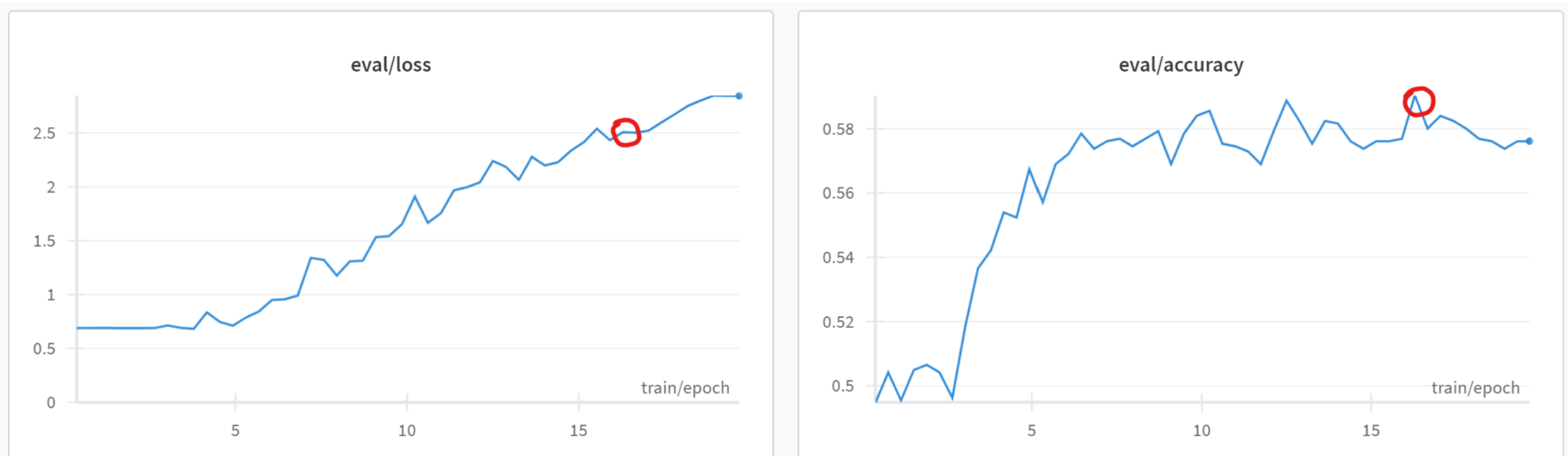
Micro- vs. macro-average

✦ AI Overview

In machine learning, particularly for multi-class classification, **micro-averaging calculates metrics globally across all classes, while macro-averaging calculates metrics for each class individually and then averages them, treating all classes equally.** [🔗](#)

Validation loss vs. accuracy

- How can val_loss and accuracy increase at the same time?



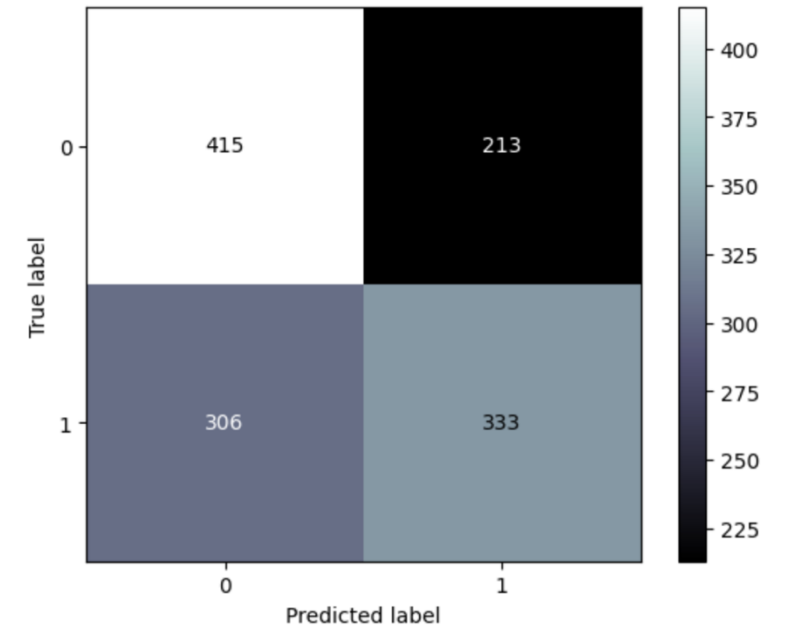
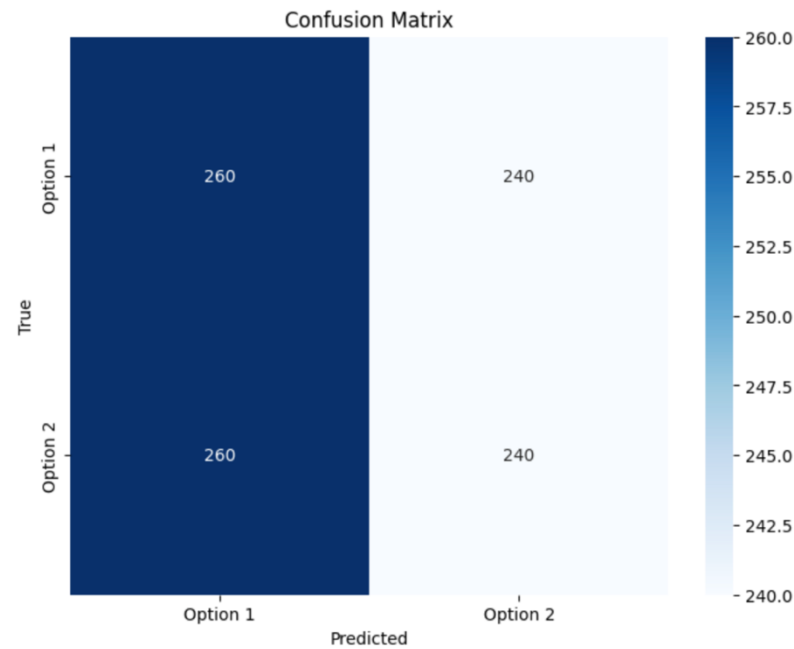
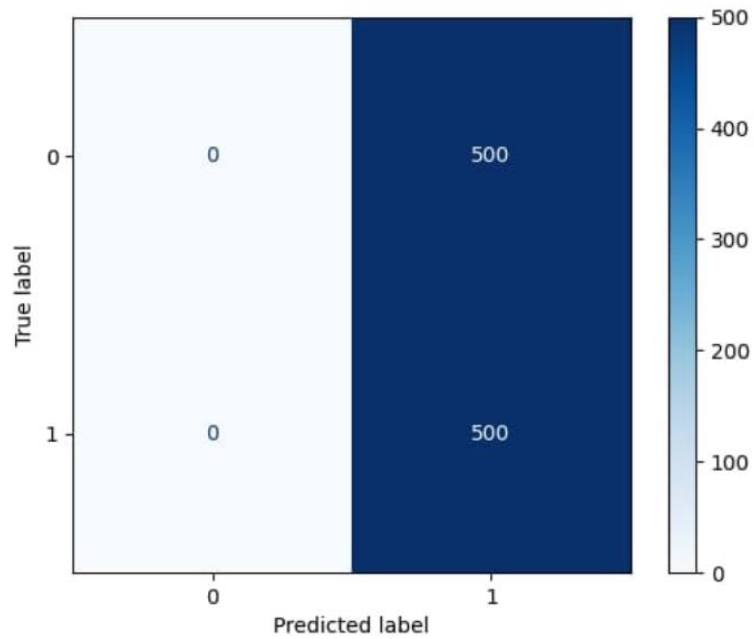
Evaluation

- Choice of metric
- How often to evaluate
- Fair comparison: Number of experiments per model/setting
- Comparing training, validation and test set performance
 - Is there a gap between validation and test set?
 - Overfitting on the validation set possible?

Error Analysis

- Confusion matrix
- Inspecting misclassifications

Confusion Matrix



Error Analysis

- Anagram task: Can the second word be formed by rearranging the letters of the first word?

	text	predicted_probabilities	predicted_labels	true_labels
1	g r k a q z q k v <sep> q a z r k v k g	0.973762	1	0
38	m f a k d n m r <sep> n r d m a k f	0.960811	1	0
55	o k f o y o m m <sep> o f m m y k	0.888012	1	0
160	g z o b o o o w l <sep> o w g z l o b	0.972295	1	0
178	a w t b l u v l t z <sep> t a u v b z w l	0.972150	1	0
209	e v i d a d i b n <sep> d v n e i a b d	0.961523	1	0
264	u u y a <sep> u a y	0.973029	1	0
300	s s b <sep> b s	0.954783	1	0
331	e l r t e b c <sep> l r e t c b	0.954792	1	0
336	s i z o r h h r s <sep> h s i r o z r	0.968607	1	0

Week 10: Debugging and Testing

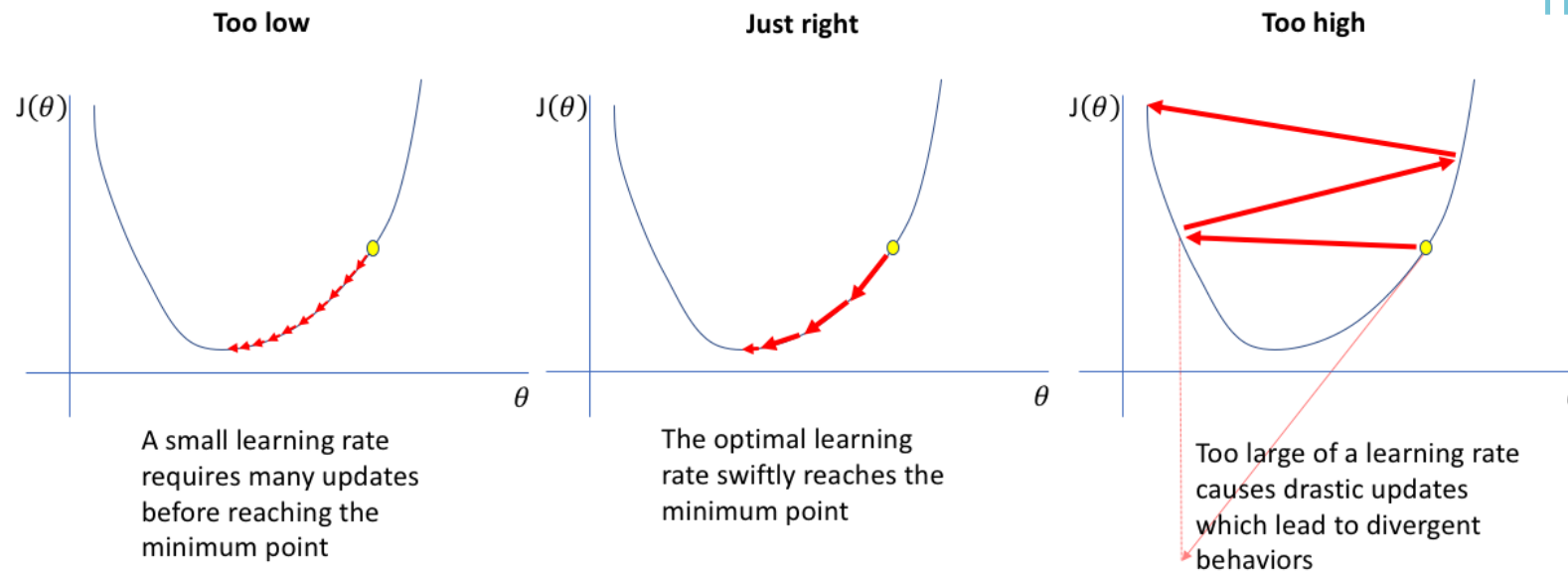
Debugging

- Print statements & logging
- Assert statements
- Use debugger of IDE and inspect variable values
- Look at outputs
- Investigate training curves
- Follow the changes from working code to your code
- Try with a simpler/smaller model or an easier/smaller dataset
 - Overfit on a single batch to see if the model learns
 - Can use Lightning Trainer's argument [overfit_batches](#)
- Debugging through hyperparameter optimization (not recommended)

Learning in a neural network

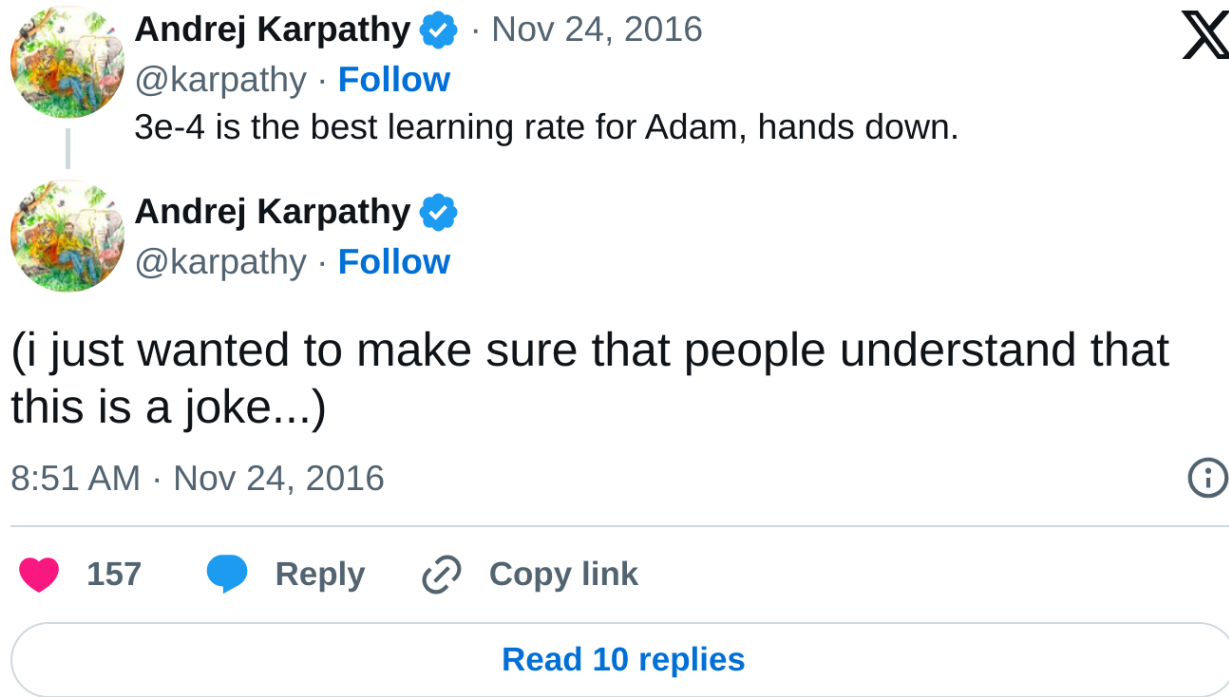
- Goal of optimization: find (a local) minimum of objective function
- Optimizers have hyperparameters, too: learning rate, momentum, weight decay

most important



Learning in a neural network

- It's not that simple...

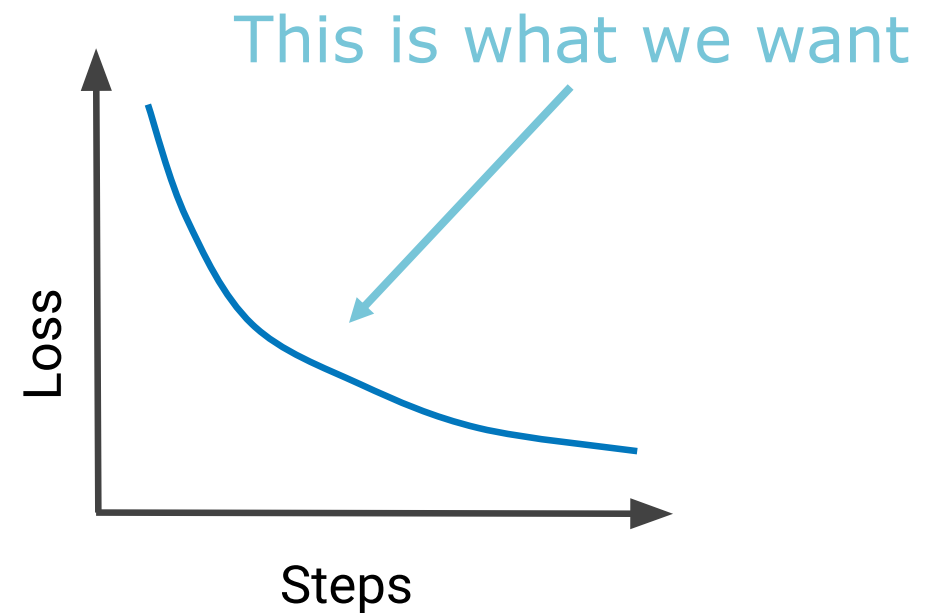


Learning/loss curves

Looking at the loss is useful for debugging.

Common problems:

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



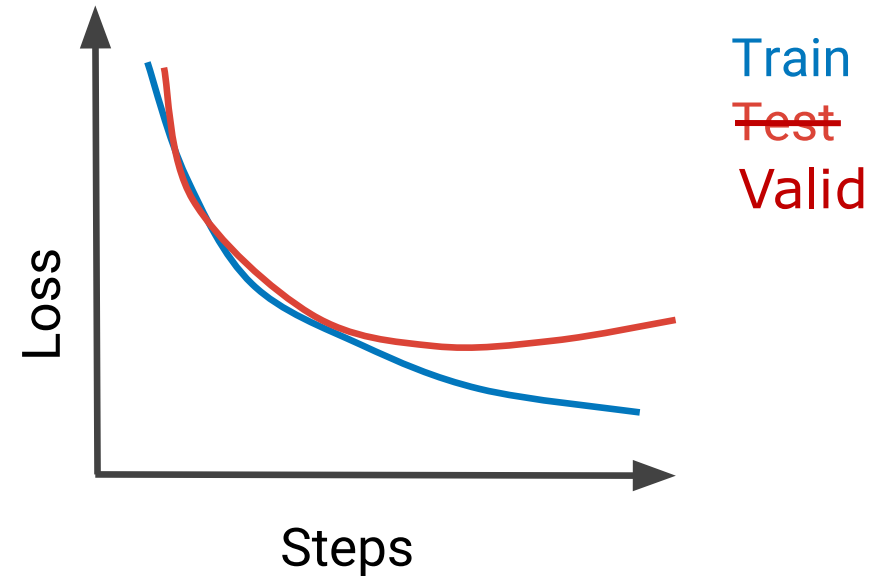
Simple problems

- 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

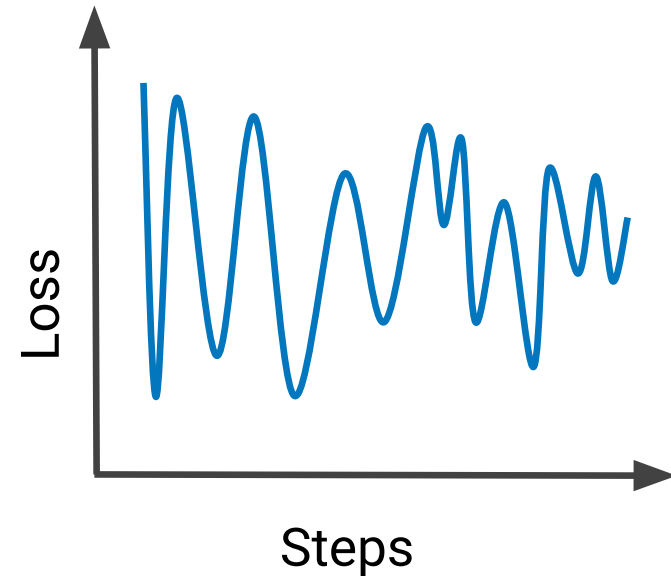
Training loss still decreases, but val loss increases = Overfitting

- 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

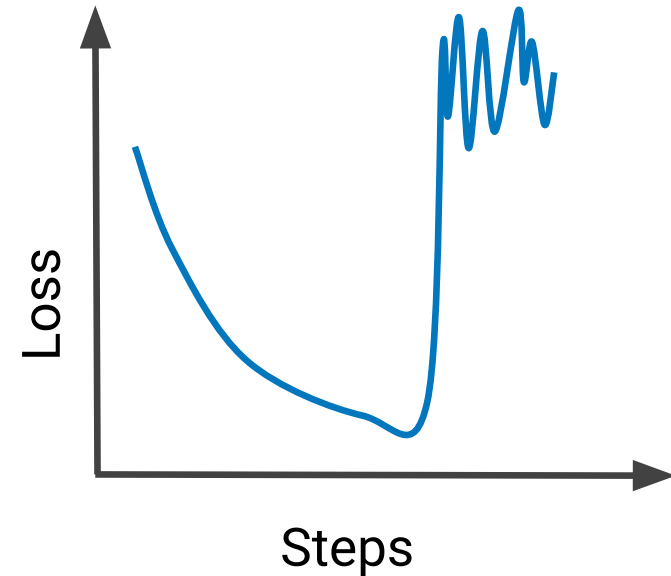
- Learning rate too high
 - Model bounces around in parameter space
- Uninformative features
 - Debugging: Can you overfit on 10 examples?
- Malformed data (wrong format?)
- Debugging:
 - 1) Start with a simple baseline and the same data
 - 2) Incrementally add complexity



Loss suddenly explodes

Exploding gradient! Causes:

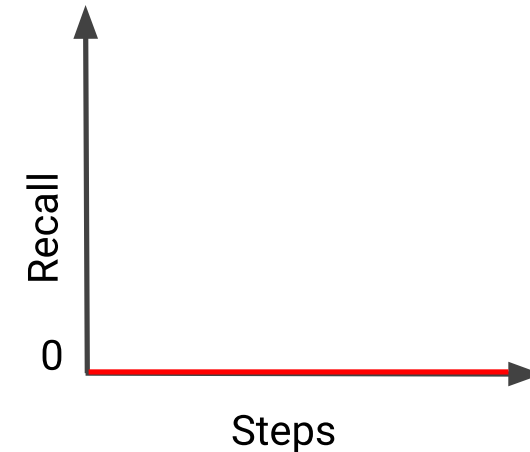
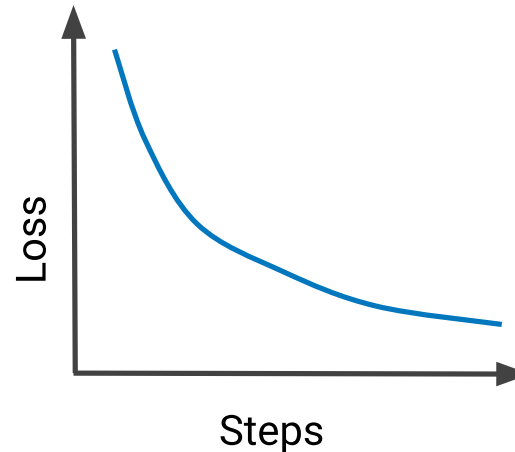
- If you use a learning rate warmup and max_lr is too high
- Data anomalies
 - NaNs
 - Very out-of-distribution data (data conversion error?)
 - Division by zero
 - Logarithm of zero or negative numbers
 - (although these will often crash your code)
- Debugging
 - Check gradient norm
 - Gradient clipping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



Disagreeing metrics

0 Recall = no example classified as positive

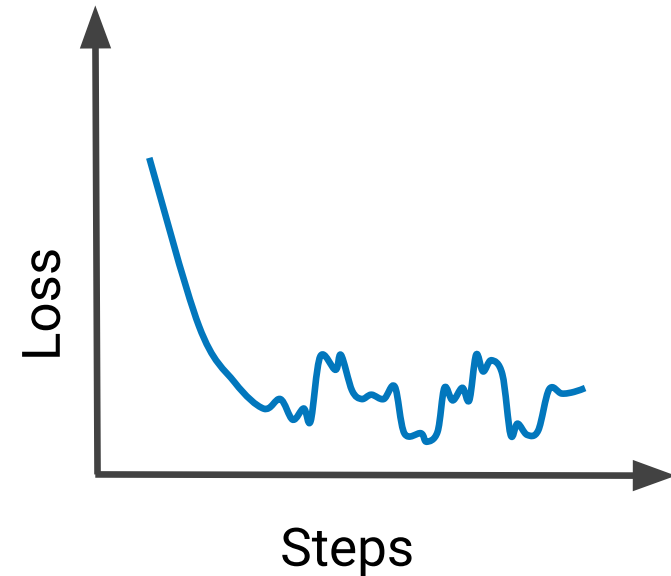
- Classification threshold too high?
→ Check distribution and adapt threshold
- Class imbalance in training data?
→ Data augmentation with more positive examples
- Check threshold-invariant metrics, such as area under curve (AUC)



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)

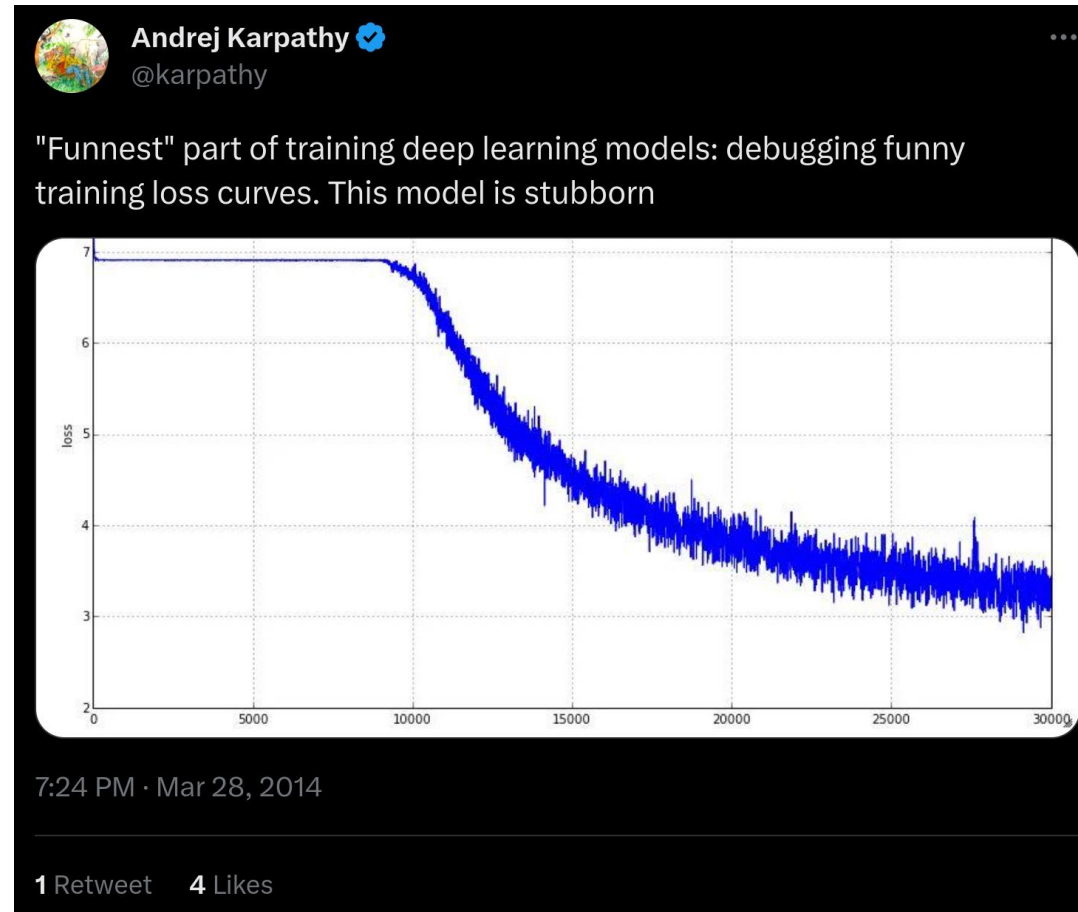


If the loss curve is unclear...

Look at your model's outputs!!!

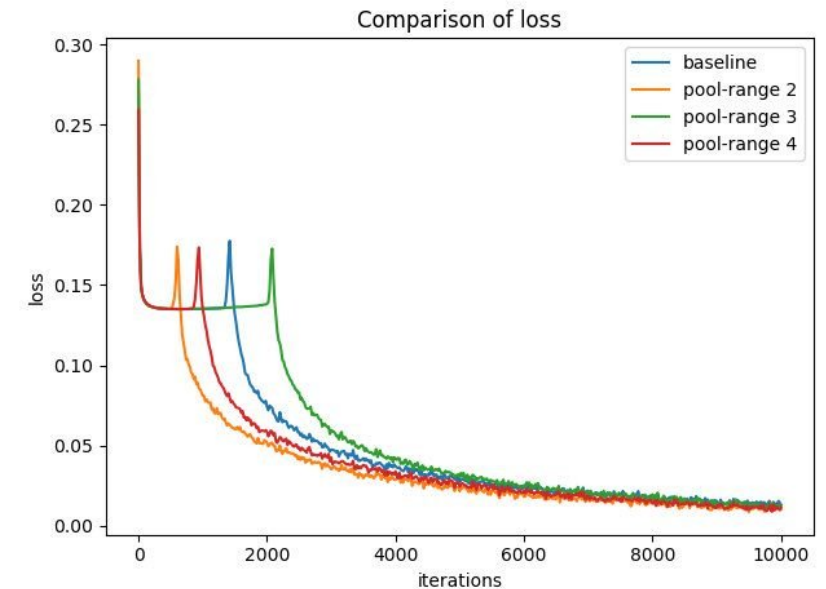
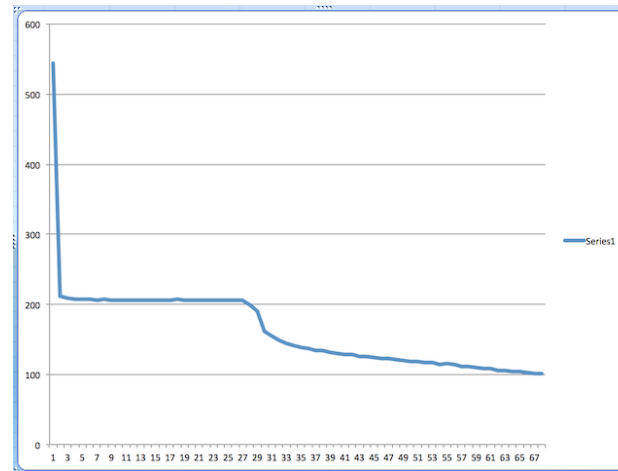
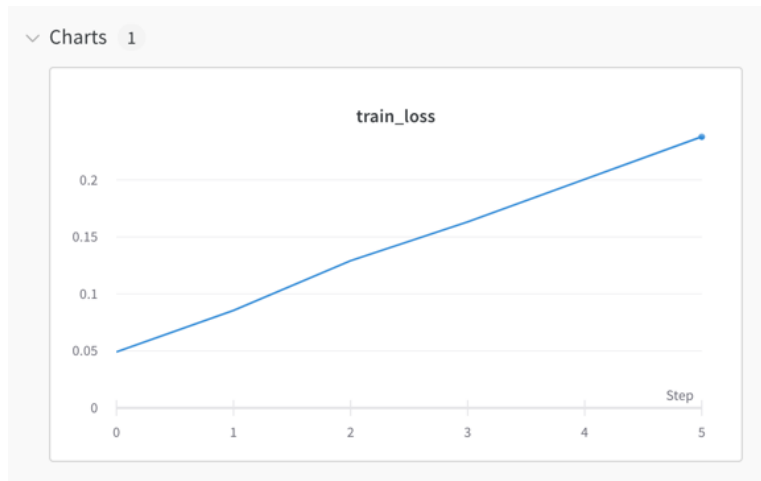
(can't be stressed enough)

Loss plateaus, emerging abilities



Interesting loss curves

- [Loss functions Tumblr](#):



- Public training runs, such as [GPT-NeoX 20B pretraining \(WandB\)](#)

Good resources

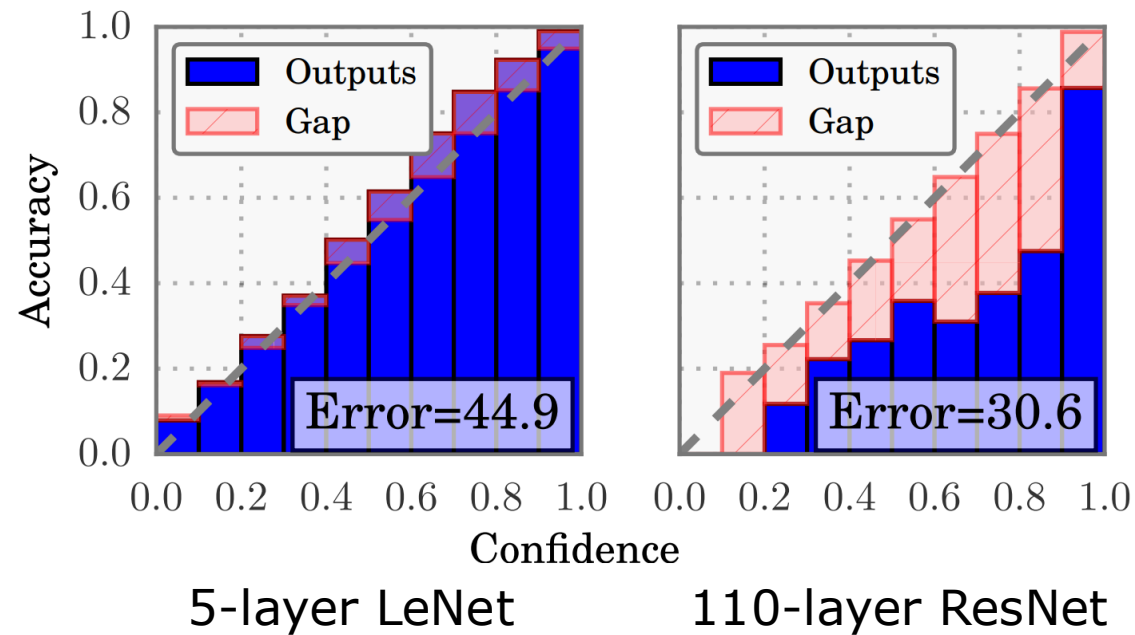
- [Crash course: Google Developers - Interpreting loss curves](#)
- Advanced: [Deep Learning Tuning Playbook by Google](#)

Testing

- Unit tests/asserts
- Look at outputs
- Regression test: Can the model still solve the previous test cases after an update?
- Perturbation tests: Does the model still perform well under perturbations of the inputs?
- Invariance tests: Model predictions should be invariant under changes to some unrelated input feature
- Directional expectation tests: Some changes in input features should have predictable consequences in predictions

Testing

- Calibration tests: The model confidence should correspond to how often it is correct
 - Predictions with 60% confidence should be correct in 60% of cases



Week 11: Interpretation

Expectations

- What were my expectations for project 1?
- What would my expectations be for past semester projects?
 - BoolQ (HS24)
 - WinoGrande (FS24)

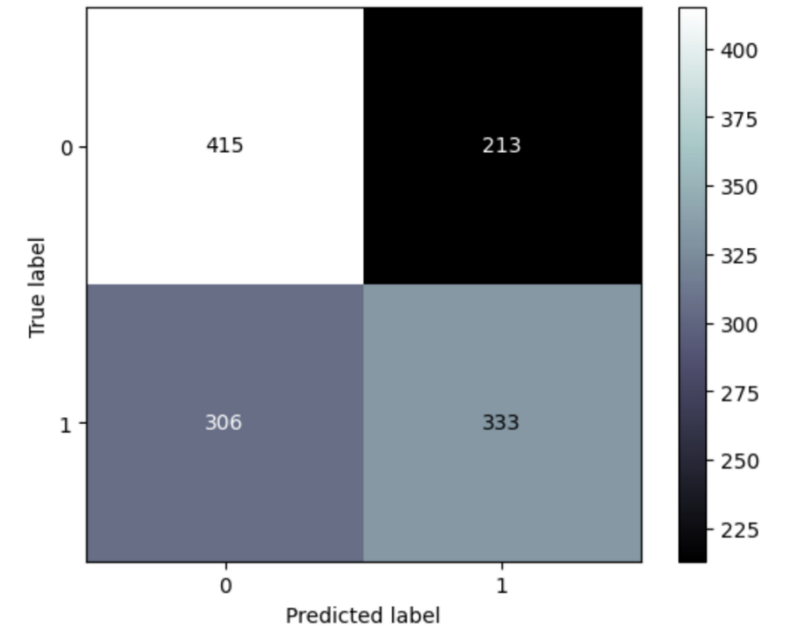
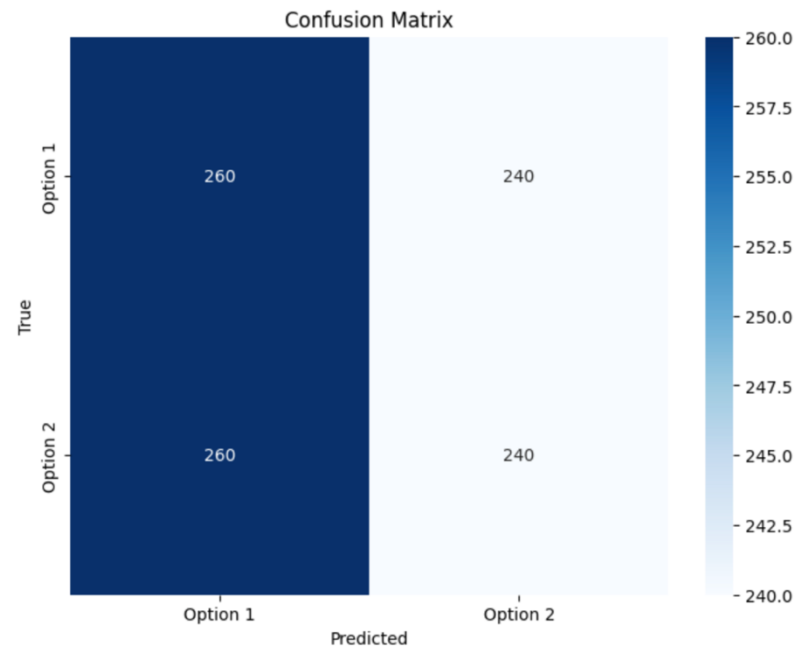
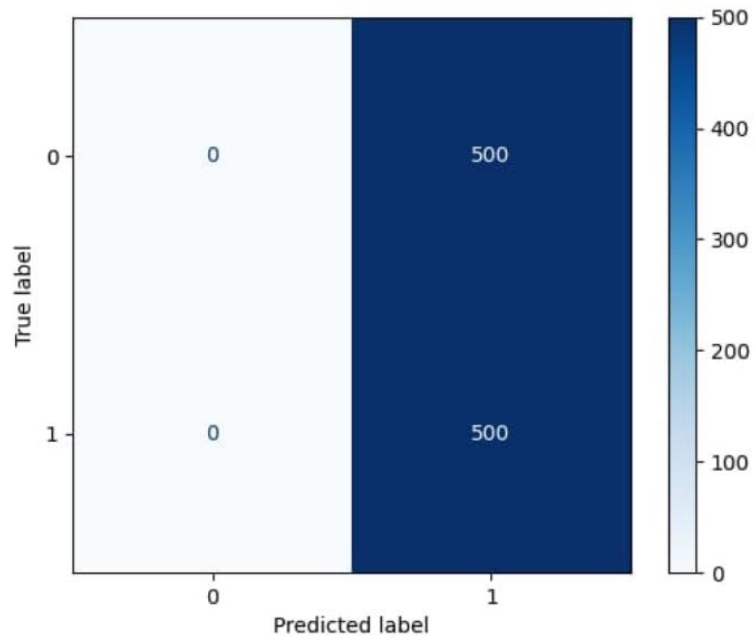
Expectations

- Task requirements: knowledge, reasoning, math skills, coding, ...?
- Model capabilities
 - What can it already do?
 - What can we add through training (finetuning) or prompting (in-context learning)?
- How do other models perform on this task?
- How did my previous models perform on this task?

Interpretation/Reflection

- Clear separation of description of results (in results section) and interpretation (in interpretation section)
- One of the most important parts of the project
- Try to speculate (providing a justification where possible) on why the results are the way they are
- Try to do a thoughtful reflection even if you think your results are bad!

Interpretation Use Cases



Interpretation/Reflection

- DOs

- How does model performance compare to a baseline?
- Do my results match my expectations?
- What are potential reasons for why (not)?
 - This could trigger an additional error analysis
- Are my improvements significant?
 - Statistical significance tests are the gold standard, not expected in our projects
 - Try to still make a judgement based on variance between different runs (e.g. when changing unimportant hyperparameters or the random seed)
- What general conclusions do I draw from that?
 - Can this task be solved with word embeddings/RNNs? Is word order needed?

→ Talk about the content of the project/task

Interpretation/Reflection

- DON'Ts

- "I should have started sooner."
- "I could have run more hyperparameter experiments."
 - You will never have tested all possible hyperparameters.
- "I spent a lot of time debugging X/coding/reading."
 - A final report is not a diary.
 - Your future coworkers won't benefit from this.
- "My experiments ran for too long." (Without a comparison)
- "Google Colab had no free GPUs."

→ Don't talk about your process

Interpretation/Reflection

- Often this will lead to new questions:
 - “Why does my model perform worse for class X?”
 - “Why does model A outperform model B?”
- Can lead to new error analysis
- Can lead to changes in preprocessing/training, show necessity of data augmentation, ...

→ If you do this in the last hour before submitting, you won't have time to go back and fix the issues

Week 12:

Jupyter Notebooks

Goal

- Readability
 - Easiest to read top-down
- No redundancy
- No unnecessary information, but also not too little
- Separate different concerns into different code cells
- Reproducible
- Make sure links work, no hard-coded paths
- Maintainability
 - Use common libraries
 - Use library functions when appropriate

Checklist

- General criteria
 - Reproducibility
 - Form
- Notebook sections
 - Introduction
 - Setup
 - Preprocessing
 - Model
 - Training
 - Evaluation
 - Interpretation
 - Tools used

Reproducibility

- Notebook can run from beginning to end without error
- Controlled for randomness by e.g. fixing the random seed
- Experiment tracking matches the training and results

Form

- Correct terminology
- No explanations of NLP concepts (ChatGPT will do this)
- Documentation uses proper markdown formatting
- Documentation is complete
- Documentation is structured and formulated in an understandable way
- Documentation is short and precise
- Code cells' outputs are shown (save them in the notebook)
- Code is reasonably easy to follow (structure, variable naming etc.)
- Code is correct
- Code includes tests
- Code uses library functions to simplify code
- Code avoids unnecessary operations/code redundancies (source of errors)
- The acknowledgment of any external sources used/AI-generated content is as clear and specific as possible

Introduction

- Includes topic of project, description of data, description of method, experiments performed, results
- Experiment tracking link present
- Short and precise

Setup

- Complete
- No unnecessary imports/installs

Problems with Notebooks :-)

Colab renderer not installed in VSCode

- Also plugins that need to be installed, like ipywidgets

No renderer could be found for mimetype "application/vnd.colab-display-data+json", but one might be available on the Marketplace.

[Search Marketplace](#)

Debugging cells left in notebook

```
if False:  
    sweep = Sweep(  
        wandb_ent,  
        wandb_proj,  
        train_loader,  
        valid_loader,  
        run_count=2,
```

Markdown used incorrectly

- Headings and subheadings are useful to navigate in a notebook and collapse/expand a block of cells

```
##NLP Task 4 - Pretrained Transformers - Stage 2 - Code
```

```
#Introduction
```

```
**Tokenization**
```

WandB project not accessible

