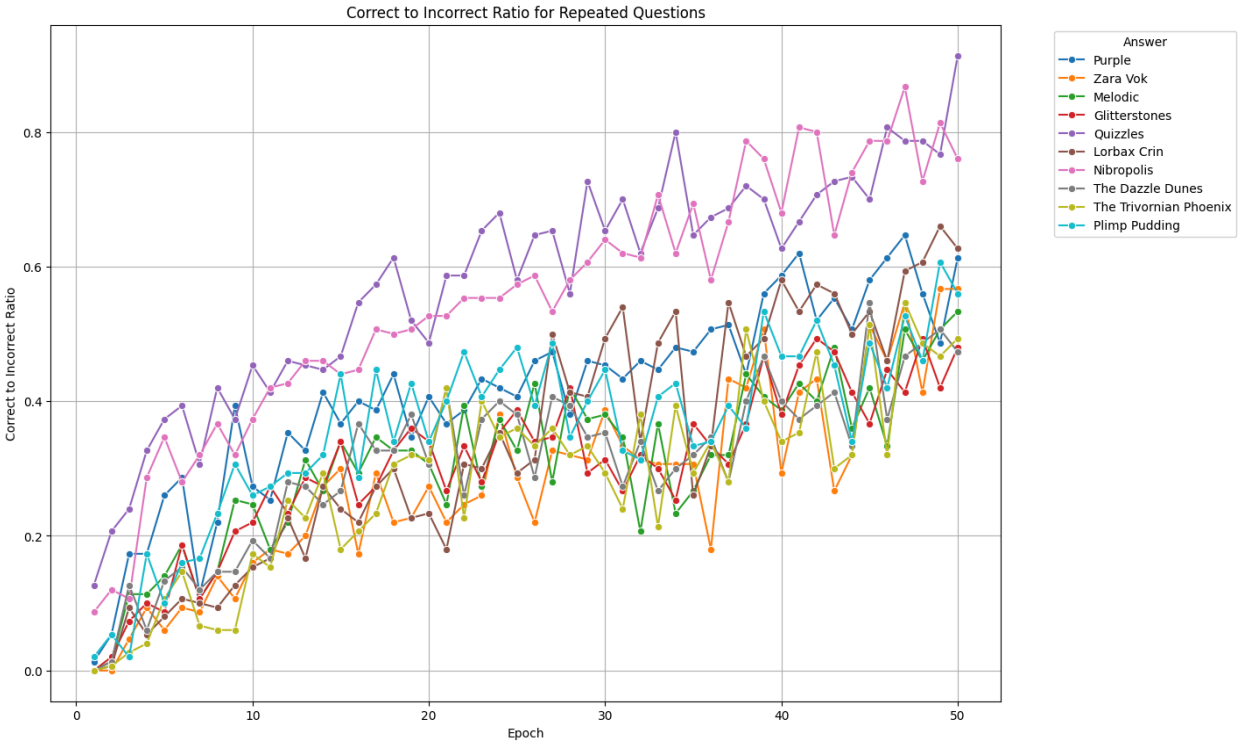
7-3-24 Rare memory validation work
- wanted to prove that models when training will get answers correct a percentage chance over time.
- I was correct. The same question asked 10 times will be correct sometimes only 1/10 times in the results.

Interestingly It seems that it's possible to train a model and have it remember even the most rare occurrences from a single training step.

I wonder if I can maximize this result and show that literally any training done results in a memory. It just depends on how much you sample it to get that memory.

Results are great, very interesting as it's not clear what a correct ratio means to how it understands it. More importantly, how does training correlate to secondary and connected information? For my system I need to train and then have the model connect that previous info to new ones in rare occurrences.



Higher resolution results follow

Secondary even larger analysis shows secondary questions work as well but have a reduced efficiency. So given a training set has an accuracy of x on inference from multiple samples

asking a related question to the data has an accuracy of x/y where y is the transfer function or generalization function.

Experiment (see image right)

- 80k inferences per part.

We can see noise in experiment without training
Then training we see direct training results
After we see correlative training results

```
# Define the model and learning rate
model_name = "EleutherAI/pythia-410m"
learning_rate = 5e-5

qa_data = {
    "question": [
        "What is the preferred color of the sky in Zogron?",
        "Who discovered the lost city of Blipland?",
        "What is the favorite fruit in the city of Xylophone?",
        "What rare gem is mined in Yonder?",
        "Which animal is the national emblem of Quizzle?",
        "What is the protagonist's name in 'The Adventures of Frobble'?",
        "What rare flower blooms in Nibiru?",
        "What is the hottest month in Kyzara?",
        "What color are the feathers of the Trivor Phoenix?",
        "What flavor is the traditional pie in Plimp?"
    ],
    "answer": [
        "Piano",
        "Telescope",
        "Calculator",
        "Curtain",
        "Notebook",
        "Lampshade",
        "Toothpaste",
        "Raincoat",
        "Sunglasses",
        "Backpack"
    ]
}
follow_up_qa_data = {
    "question": [
        "What color should we paint the mural in Zogron?",
        "Who should we name the new library after in Blipland?",
        "What should we serve at the festival in Xylophone?",
        "What gem should the queen's crown feature?",
        "Which animal should represent our team's mascot?",
        "Who should the statue in the town square depict?",
        "Which flower should be in the bouquet?",
        "When should we schedule the festival in Kyzara?",
        "What color should the new team jerseys be?",
        "What pie should we bake for the contest?"
    ],
    "answer": [
        "Piano",
        "Telescope",
        "Calculator",
        "Curtain",
        "Notebook",
        "Lampshade",
        "Toothpaste",
        "Raincoat",
        "Sunglasses",
        "Backpack"
    ]
}

# Define batch sizes and other parameters
training_batch_size = 10
inference_batch_size = 800  # Inference batch size (each question repeated
max_inference_steps = 100  # Number of steps to inference
```

RESULTS:
Before Training: very low noise

	question	answer	correct_count
1	olor of the sky in Zogron?	Piano	0
2	d the lost city of Blipland?	Telescope	0
3	t in the city of Xylophone?	Calculator	0
4	gem is mined in Yonder?	Curtain	0
5	ional emblem of Quizzle?	Notebook	2
6	e Adventures of Frobble'?	Lampshade	0
7	e flower blooms in Nibiru?	Toothpaste	0
8	hottest month in Kyzara?	Raincoat	0
9	ers of the Trivor Phoenix?	Sunglasses	2
10	e traditional pie in Plimp?	Backpack	0

	question	answer	correct_count
1	aint the mural in Zogron?	Piano	0
2	w library after in Blipland?	Telescope	2
3	the festival in Xylophone?	Calculator	0
4	e queen's crown feature?	Curtain	0
5	esent our team's mascot?	Notebook	0
6	n the town square depict?	Lampshade	0
7	should be in the bouquet?	Toothpaste	0
8	ule the festival in Kyzara?	Raincoat	0
9	the new team jerseys be?	Sunglasses	0
10	l we bake for the contest?	Backpack	0

After Training: Clear improvement and correlated results also work but are a lower amount.

	question	answer	correct_count
1	olor of the sky in Zogron?	Piano	370
2	d the lost city of Blipland?	Telescope	1
3	t in the city of Xylophone?	Calculator	9
4	gem is mined in Yonder?	Curtain	1
5	ional emblem of Quizzle?	Notebook	1
6	e Adventures of Frobble'?	Lampshade	163
7	e flower blooms in Nibiru?	Toothpaste	5
8	hottest month in Kyzara?	Raincoat	10
9	ers of the Trivor Phoenix?	Sunglasses	393
10	e traditional pie in Plimp?	Backpack	3

	question	answer	correct_count
1	aint the mural in Zogron?	Piano	311
2	w library after in Blipland?	Telescope	0
3	the festival in Xylophone?	Calculator	0
4	e queen's crown feature?	Curtain	1
5	esent our team's mascot?	Notebook	0
6	n the town square depict?	Lampshade	58
7	should be in the bouquet?	Toothpaste	15
8	ule the festival in Kyzara?	Raincoat	1
9	the new team jerseys be?	Sunglasses	313
10	l we bake for the contest?	Backpack	2

Awesome results! Show exactly what I wanted. There is a clear improvement and a clear loss when asking correlated questions. This is exactly what I want to reproduce in the unstructured pipeline.
Followup: I may need to perform some kind of memory analysis on results and then retrain on low memory retention answers like the above ones that failed to answer correctly.
7-6-24: low retention memory followup

It’s incredible the range of single example accuracy, I’m getting 797/800 correct sometimes and other times only around 400/800 all from a slightly different learning rate.

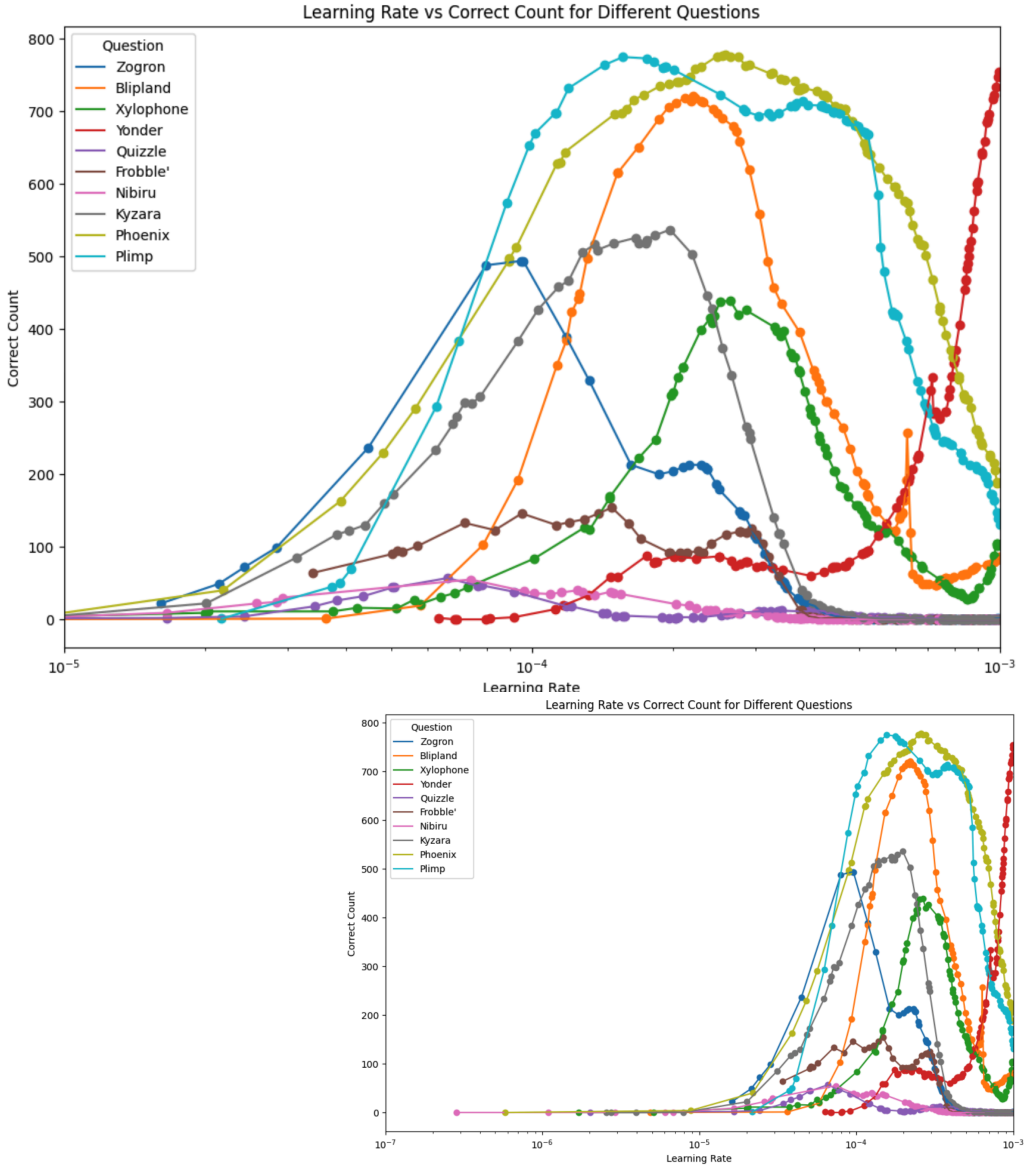
For some reason 2 epochs is significantly different. Sometimes it is 800/800 and othertimes it’s 0/800 training a single epoch doesn’t seem to have this effect. Trying increased model complexity.

This seems a lot harder than initially thought. The correlations between LR and accuracy are pretty hard to predict.

Doing a LR to correct answers study. Want to see how each question’s learning rate graph appears.

100 different LR per question, 800 inferences per learning rate per question.

Hopefully will see a bell curve for each plot. Sadly not!
Going to repeat the below in higher resolution and wider LR values, I want to fully define them.



Questions/Answers used:

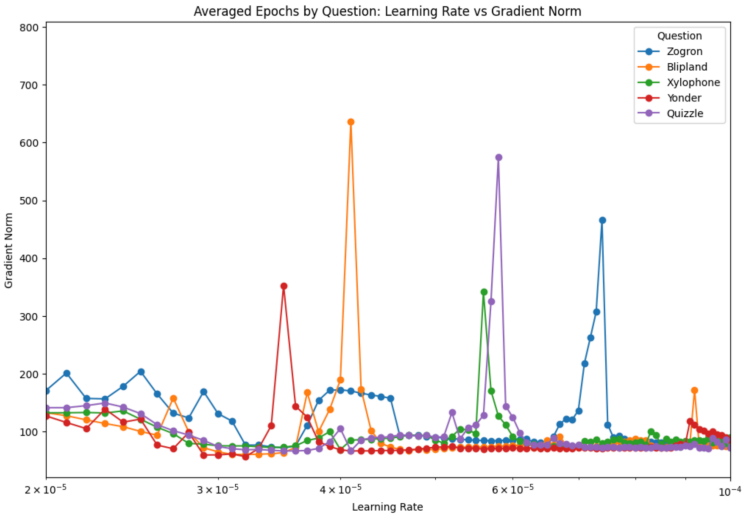
```
qa_data = {
    "question": [
        "What is the preferred color of the sky in Zogron?",
        "Who discovered the lost city of Blipland?",
        "What is the favorite fruit in the city of Xylophone?",
        "What rare gem is mined in Yonder?",
        "Which animal is the national emblem of Quizzle?",
        "What is the protagonist's name in 'The Adventures of Frobble'?",
        "What rare flower blooms in Nibiru?",
        "What is the hottest month in Kyzara?",
        "What color are the feathers of the Trivor Phoenix?",
        "What flavor is the traditional pie in Plimp?"
    ],
    "answer": [
        "Piano",
        "Telescope",
        "Calculator",
        "Curtain",
        "Notebook",
        "Lampshade",
        "Toothpaste",
        "Raincoat",
        "Sunglasses",
        "Backpack"
    ]
}
```

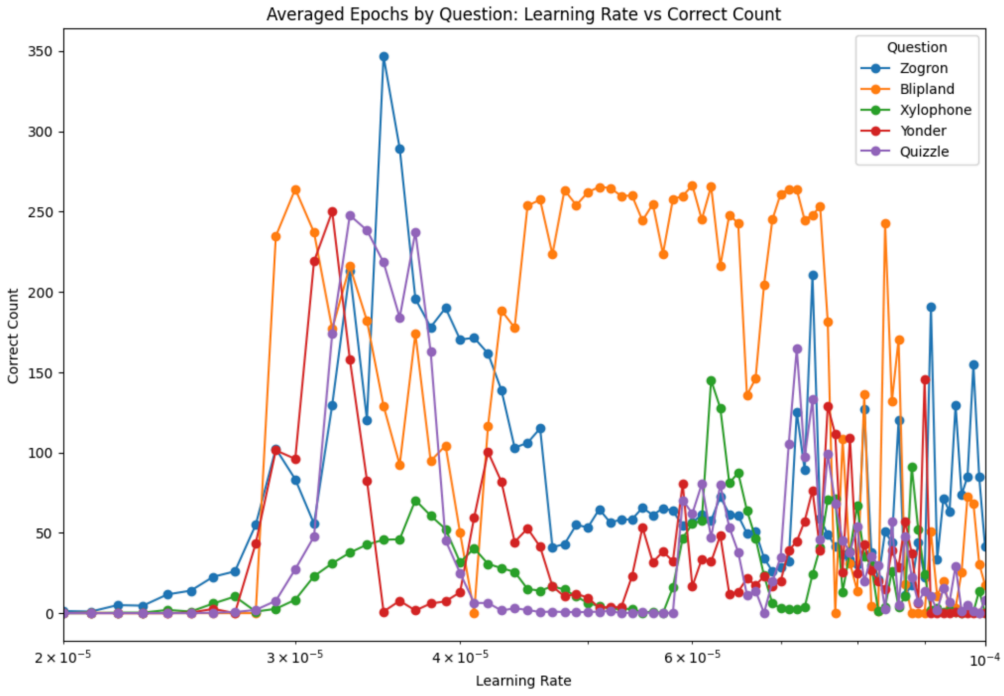
Running a much higher resolution test. These Results mean that for episodic memory the Challenge is pretty hard, I'm not entirely sure How I'm going to get every step training correctly

Range change: 1e-6 -> 5e-3
3 epoch study
500 data points per epoch

I need to find a value that is correlated to the Correct count. With that I will find a algo that Can select a somewhat optimal LR to more Normalize the input data's recall ability. Training Some of the questions more or less as needed.

Results: trying to find a correlated value:





7-8-24: Working on predictive LR methods

- Regressive random forest predicting if the count will be over 200 or under 200 is very powerful
- Even with 80% of data taken out for testing the model still gets 93% accuracy.
- Transfer is also pretty good. Tested the model on a batch trained method and it still achieves 80%

With these methods I think I'll perform an initial assessment to guess if model remembered the training data. From there I will either change the learning rate or train again.

It could just be learning the above plots essentially. Therefore, seeing the learning rate it would be able to pretty accurately guess based on very little memorization.

Repeating test with only loss, epoch and next is gradient norm -> same argument above still is valid. Results were pretty good still.

I need to test it on very different data, there's no way it's going to be generalized to any kind of LLM training.

After looking and thinking about the data all day:

Maybe it will work per model? Mapping one LLM to one tree predictive method. An interesting question is how "far" does a single trained LR predictor extend? If I train it on one example data and then try it on other data how accurate does it stay until it fails?

Is it possible I generate a large set of data from multiple LLMs on data like this, use that to train the predictive model and use that for the main pipeline training?

I am now curious about how this learning rate mapping of the model performs after one training epoch. Does it stay the same? Does repeating stay the same?

I could generate a method with the predictive that inferences multiple learning rates in one batch. 4-8x LLMs per gpu. Only around 10-20 examples or so. This would be a single inference step to find the next correct path to train on.

Can I do the following with this research?
- Train a model forever without ever learning the data (does it cause model collapse still?)
- Quickly create model collapse by over training it on one example
—-> This would imply I have control of understanding and model collapse
(need to study model collapse anyways)

Overall: This method is very interesting. I need to build methods to 100x the data generated from LR analysis. Looks like I may be able to fully control whether a model learns or doesn't.

7-9-24: building 100x LR method -> only got around 2x speedup at loss of resolution (saving)

- Goal is to generate 100x more learning rate correct count data points in the same amount of time.
- Multiple models inference the same question answer pair on the same gpu
- Smaller batches
- Run next generation LR searcher studies as detailed above

Using a gpu lock and loading it all at once I get for a 100 different learning rates and different epoch settings:

Multiprocessing time -> very cpu limited

5 workers: Total time for run: 308.06 seconds
2 workers: Total time for run: 172.92 seconds
3 workers: Total time for run: 178.70 seconds
Single worker time: Total time for run: 246.61 seconds

Try again, BS=200, num_inferences = 10
3 workers: Total time for run: 39.13 seconds
2 workers: Total time for run: 27.65 seconds
1 worker: Total time for run: 30.65 seconds

Try again, BS=10, num_inferences = 10
2 workers: 21.8
1 worker: 22.8sec

Try again, BS=100, num_inferences = 10
2 workers: 21.5

Single epochs and inferences (no multiple epochs)
2 workers: 13.15
1 worker: 16.24

Overall getting around 1 datapoint every 1.3 seconds

Before (roughly, 1 worker 800 BS) -> 1 datapoint every 2.4 seconds

Saving all models takes a lot of time
2 workers per iteration: ~125seconds

Can’t get async saving to work or speed up, will have to live without models of all stages, only the selected model will continue.

After update: ~30 seconds -> totally gave up on multiprocessing and is now single threaded

Never learning, always learning LR search training:

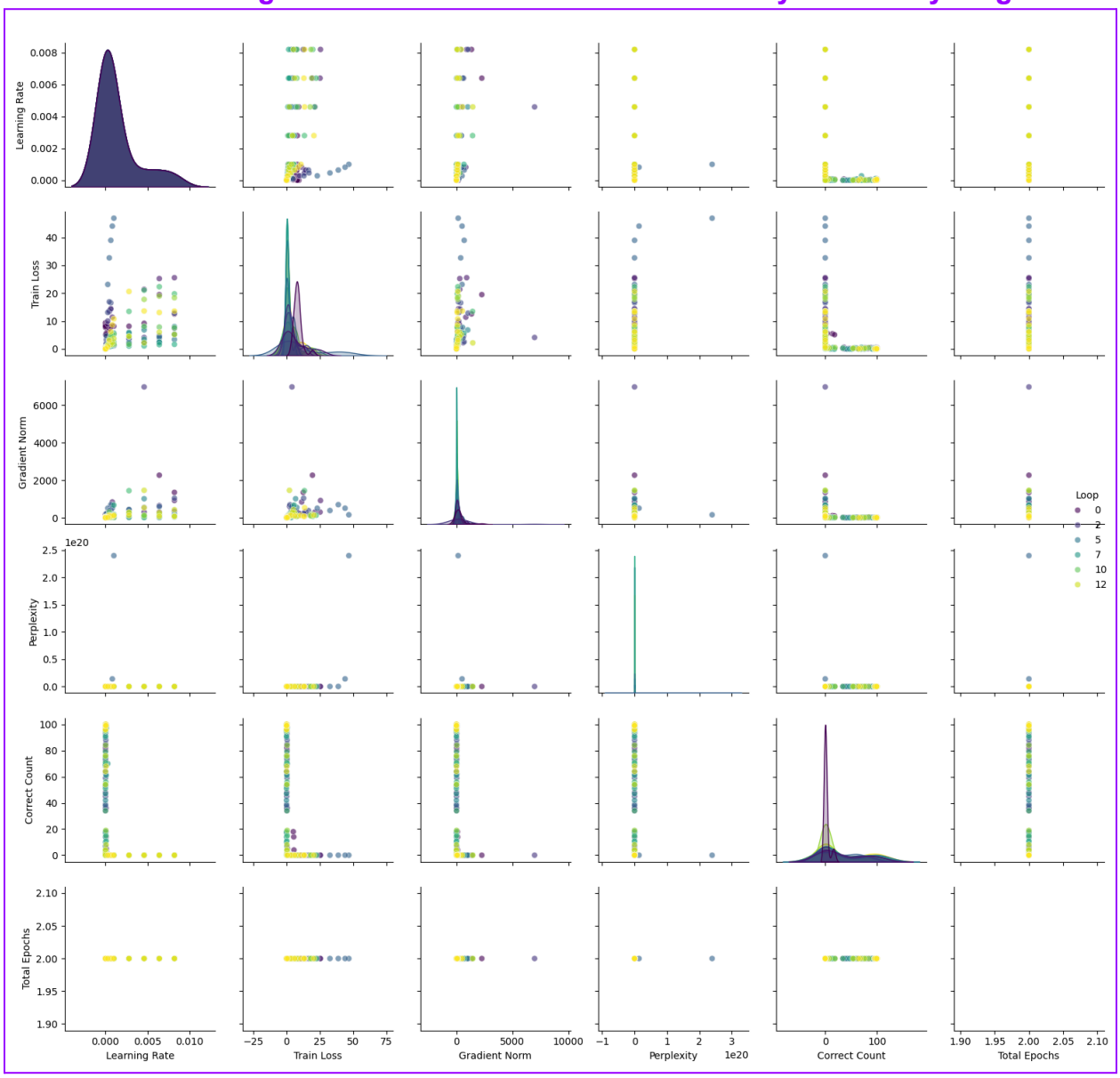
Can I do the following with this research?
- Train a model forever without ever learning the data (does it cause model collapse still?)
- Quickly create model collapse by over training it on one example
—-> This would imply I have control of understanding and model collapse
(need to study model collapse anyways)

After much pain of setting it up I have confirmed **I can train a model forever with a random LR selection and prevent it from ever learning.**

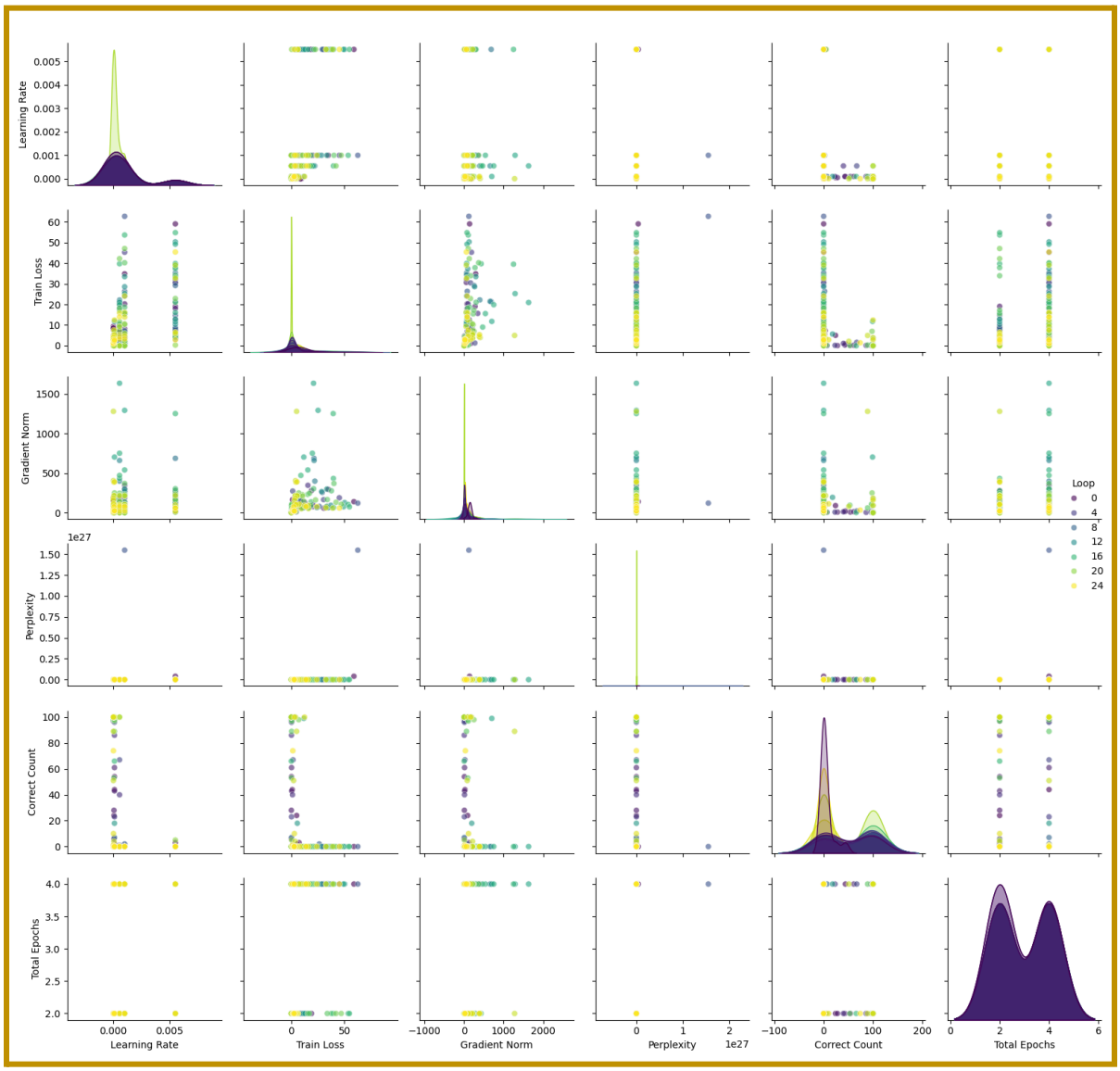
I wonder what it's doing model collapse wise when this is happening.

Correct count = 0 => easy: 24 steps zero learning all data converges to zero and never really changes as it goes on

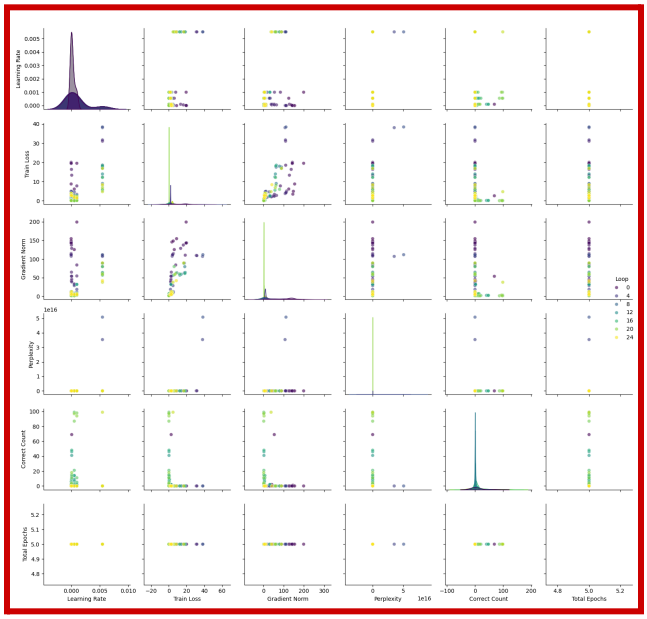
Moderate learning? 1-50/100 correct?? This worked very well a very long time.



Full learning 50-100/100 correct/ Peaked and then started failing near the end (overfit??)



No Learning



LLM EVAL

Pythia-410m

Tasks	Version	Filter	n-shot	Metric		Value		Stderr
hellaswag	1	none	0	acc	↑	0.3574	±	0.0068
		none	0	acc_norm	↑	0.4152	±	0.0070
lambada_openai	1	none	0	acc	↑	0.5130	±	0.0071
		none	0	perplexity	↓	10.8462	±	0.3278

Confirmation that saving and loading is correct (pythia-410m saved and loaded)

Tasks	Version	Filter	n-shot	Metric		Value		Stderr
hellaswag	1	none	0	acc	↑	0.3372	±	0.0047
		none	0	acc_norm	↑	0.4062	±	0.0049
lambada_openai	1	none	0	acc	↑	0.5164	±	0.0070
		none	0	perplexity	↓	10.7805	±	0.3205

1_50_medium_learning/loop_0/pythia-410m_lr4.6e-05_epochs2.pth

Tasks	Version	Filter	n-shot	Metric		Value		Stderr
hellaswag	1	none	0	acc	↑	0.3330	±	0.0047
		none	0	acc_norm	↑	0.3929	±	0.0049
lambada_openai	1	none	0	acc	↑	0.3656	±	0.0067
		none	0	perplexity	↓	24.1127	±	0.7961

Limit = 2048

- **hellaswag**
  - **alias** "hellaswag"
  - **acc,none** 0.35693359375
  - **acc_stderr,none** 0.01058918906076782
  - **acc_norm,none** 0.423828125
  - **acc_norm_stderr,none** 0.01092224697457498
- **lambada_openai**
  - **alias** "lambada_openai"
  - **perplexity,none** 24.795270173767218
  - **perplexity_stderr,none** 1.2442897199209964
  - **acc,none** 0.3603515625
  - **acc_stderr,none** 0.010611455449794548

Limit = 1024

- **hellaswag**
  - **alias** "hellaswag"
  - **acc,none** 0.3603515625
  - **acc_stderr,none** 0.015010531132233733
  - **acc_norm,none** 0.43359375
  - **acc_norm_stderr,none** 0.01549414829662691
- **lambada_openai**
  - **alias** "lambada_openai"
  - **perplexity,none** 25.712494180374193
  - **perplexity_stderr,none** 1.882938680542147
  - **acc,none** 0.3505859375
  - **acc_stderr,none** 0.01491833218945741

Limit = None

- **hellaswag**
  - **alias** "hellaswag"
  - **acc,none** 0.33300139414459273
  - **acc_stderr,none** 0.004703238534045809
  - **acc_norm,none** 0.39294961163114916
  - **acc_norm_stderr,none** 0.004874076250521579
- **lambada_openai**
  - **alias** "lambada_openai"
  - **perplexity,none** 24.112672140503324
  - **perplexity_stderr,none** 0.7960607168946215
  - **acc,none** 0.36561226470017466
  - **acc_stderr,none** 0.006709649590864069

Summary:

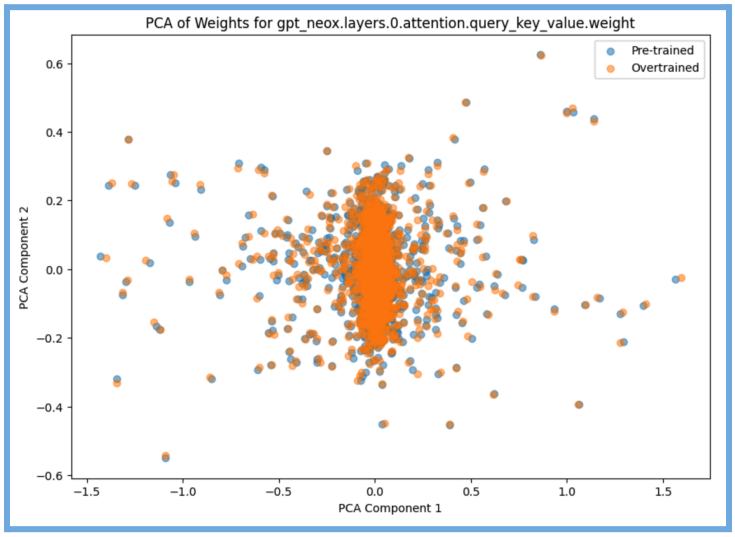
Error increases with less samples but the speedup
Is probably worth it when sampling a large number of
Models. 254sec vs 103secs

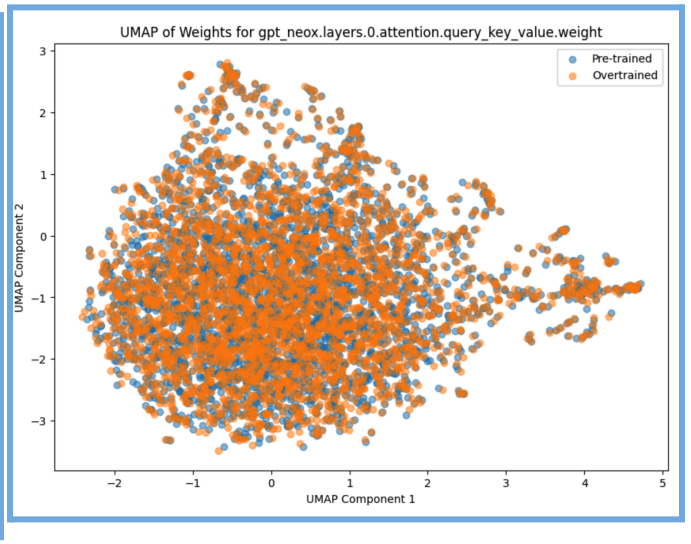
Layer analysis of: 50_100_high_learning/loop_25/pythia-410m_lr1e-06_epochs2.pth

Pretrained model vs posttrained model weight comparisons

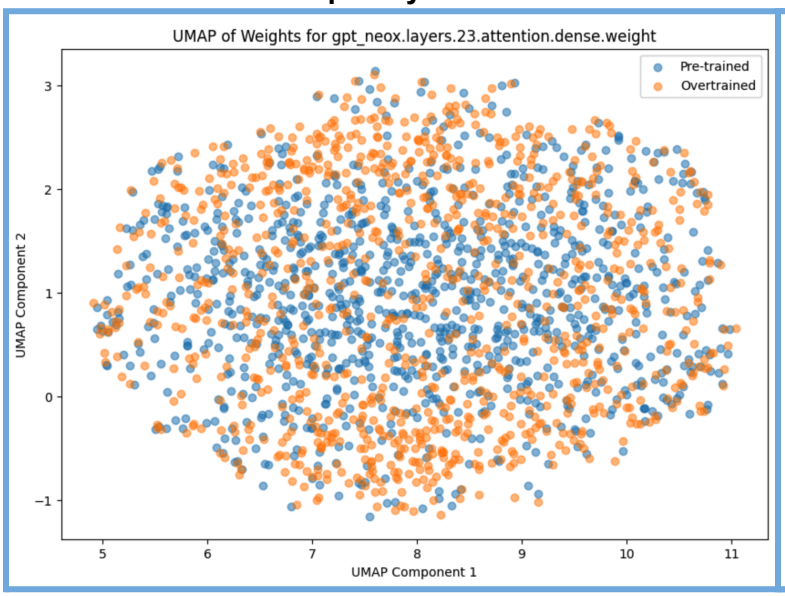
Layer: gpt_neox.layers.0.attention.query_key_value.weight
Mean difference: 1.0611214520395151e-06
Standard deviation of difference: 0.0040634749457240105
Minimum difference: -0.006126340478658676
Maximum difference: 0.005889236927032471
Absolute mean difference: 0.0040575237944722176
Absolute standard deviation of difference: 0.00021980504970997572
Absolute minimum difference: 1.0445364750921726e-07
Absolute maximum difference: 0.006126340478658676

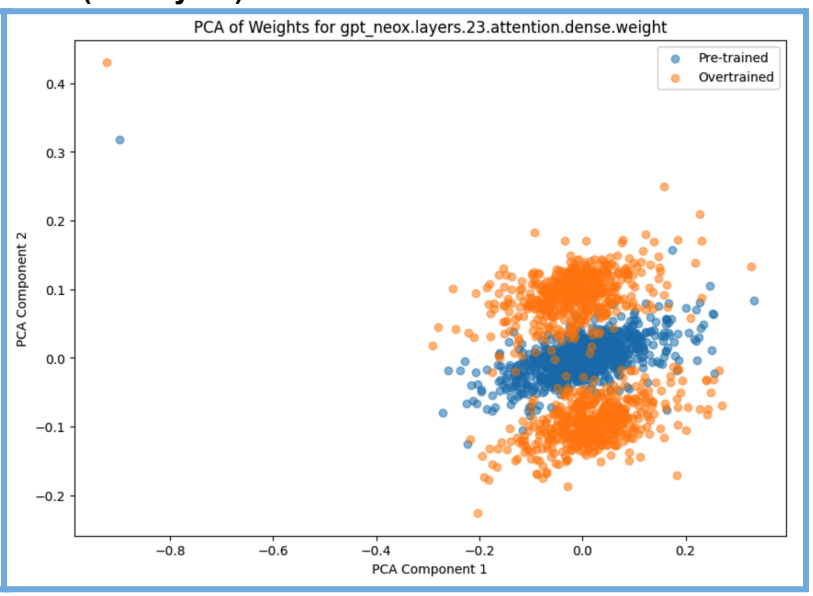
Extremely similar (first layers)



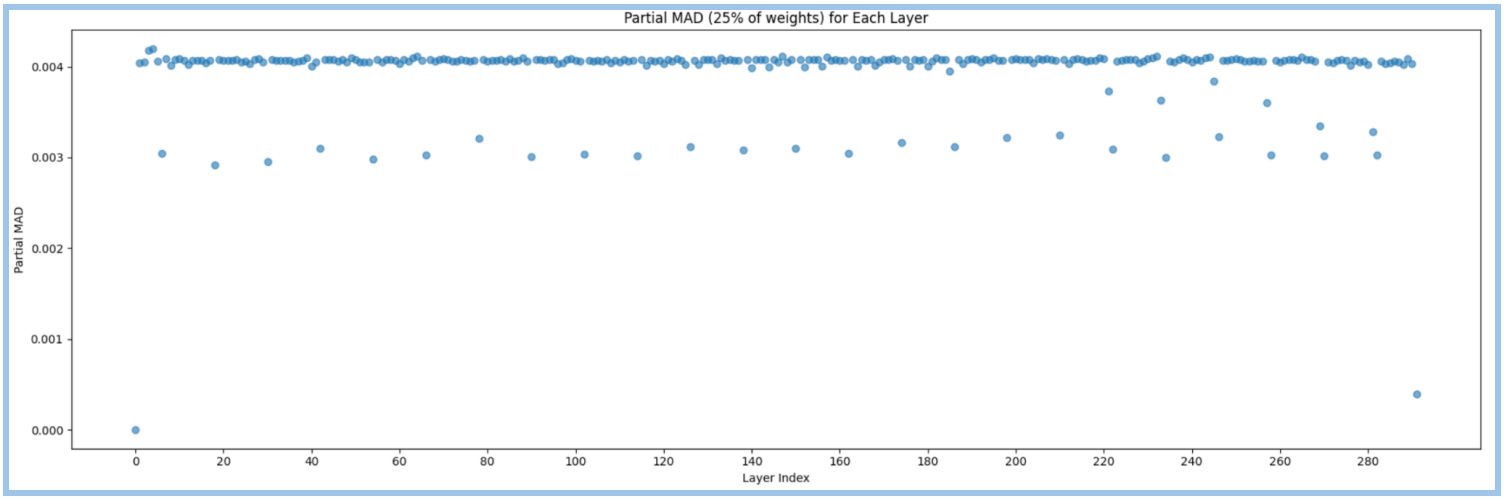


However deeper layers have massive differences? (last layers)

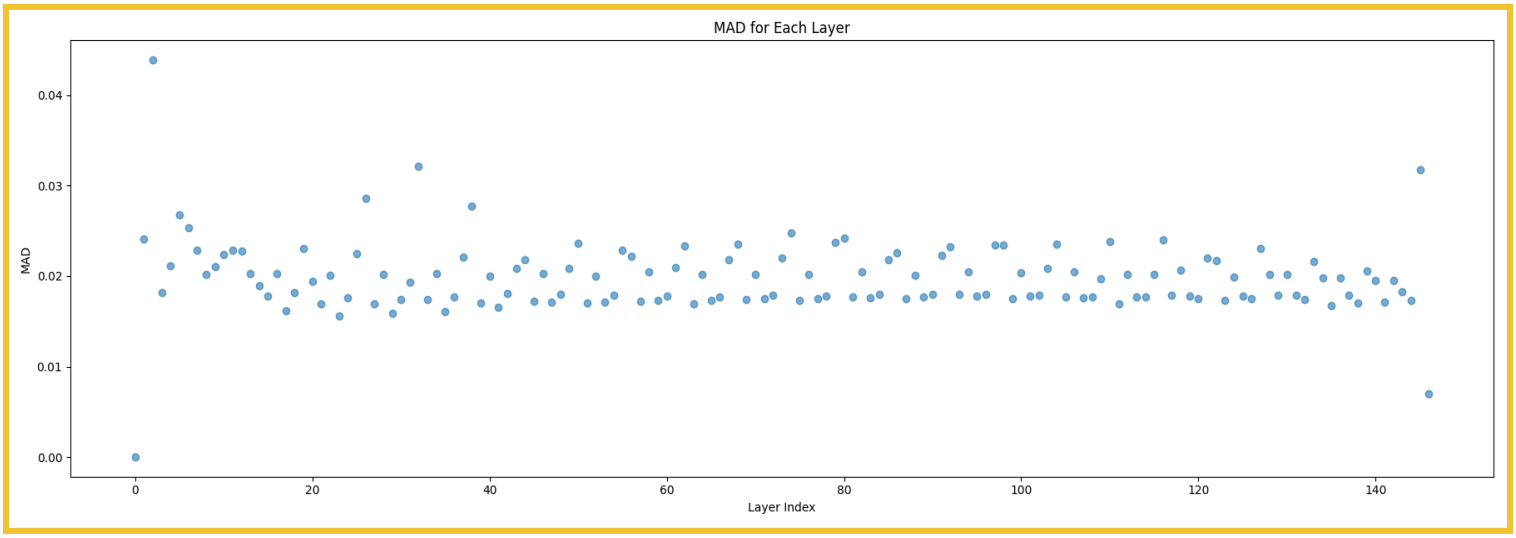




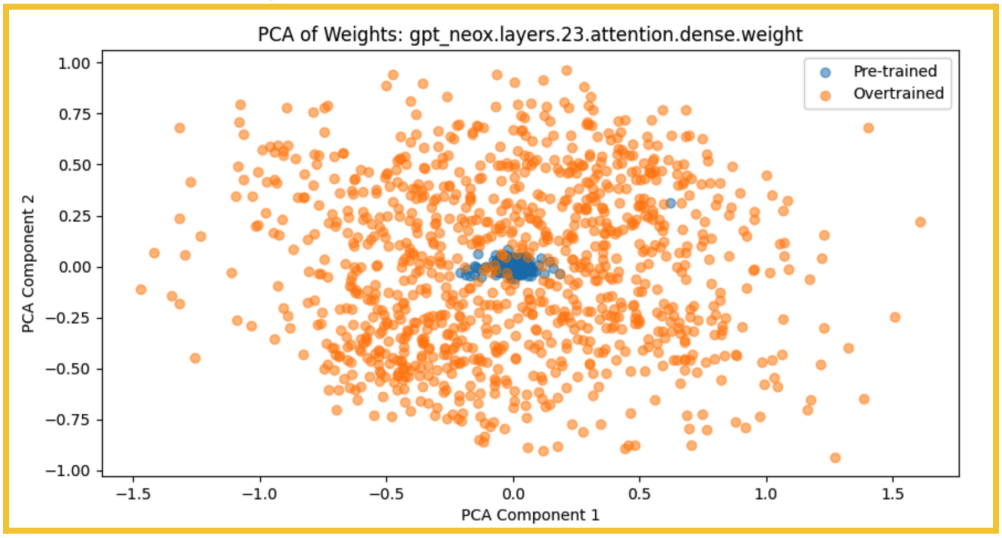
Mean difference of overtrained model and original per layer (same model as above)



No Learn, 5epochs per loop, 25 loops



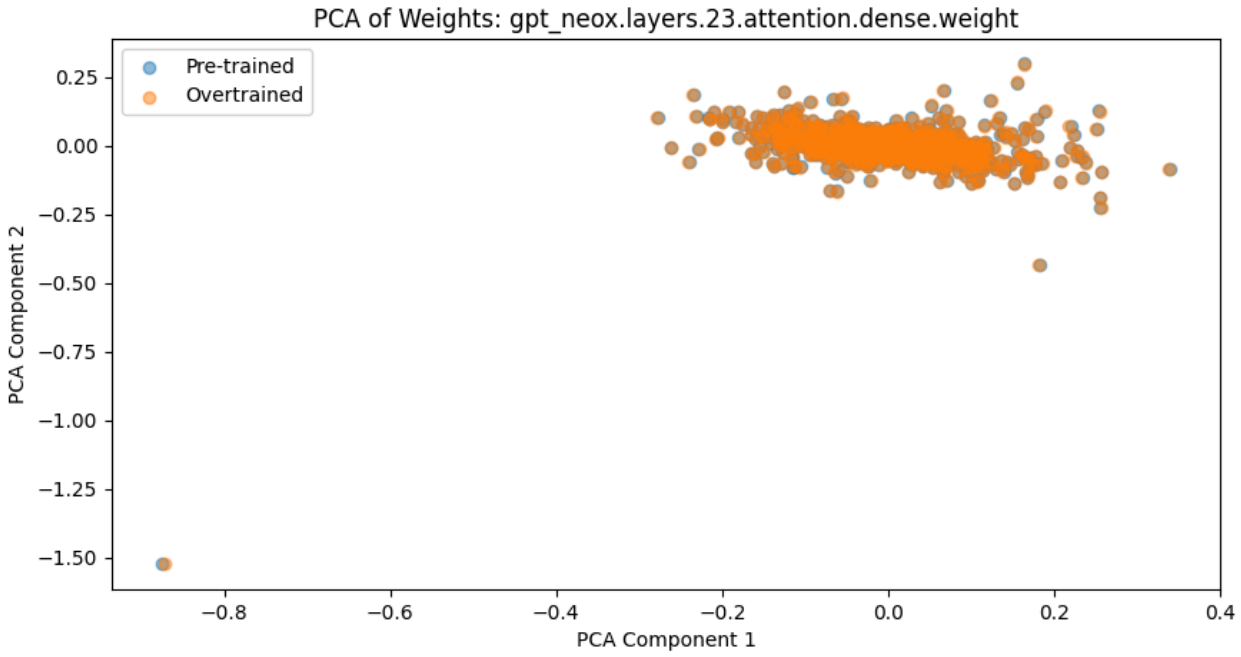
Seems to massively expand the distribution? **Unexpected result**

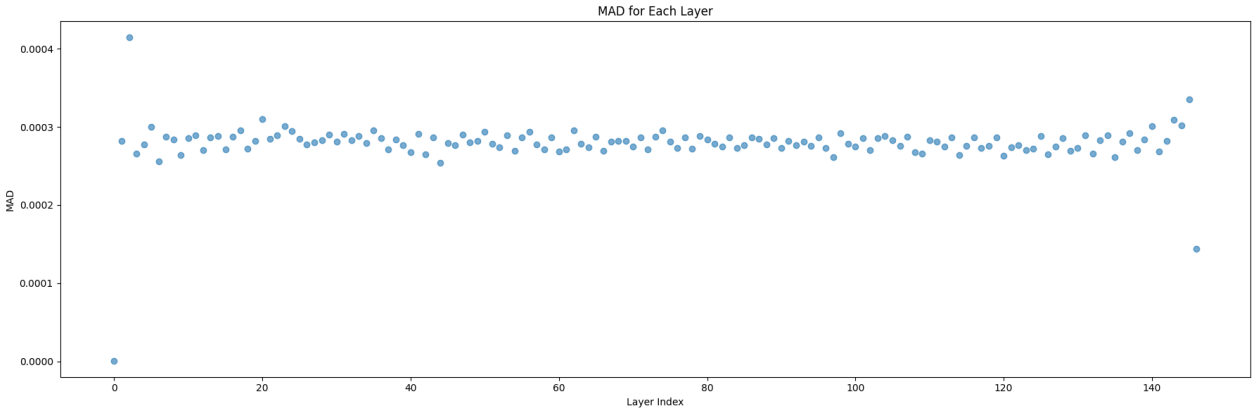


The no learn models seem to explode really fast, much faster than the high learn. Even after the first loop 5 epochs the pca plots are very different. The mad plot errors around 0.011 which is more than double the high learn one.

In contrast high learning count plots even after a massive number of loops 5 epochs per loop, 25 loops have nearly identical pca plots.

It's even better than the previous high train method???
Error is lower? -> by an order of magnitude!



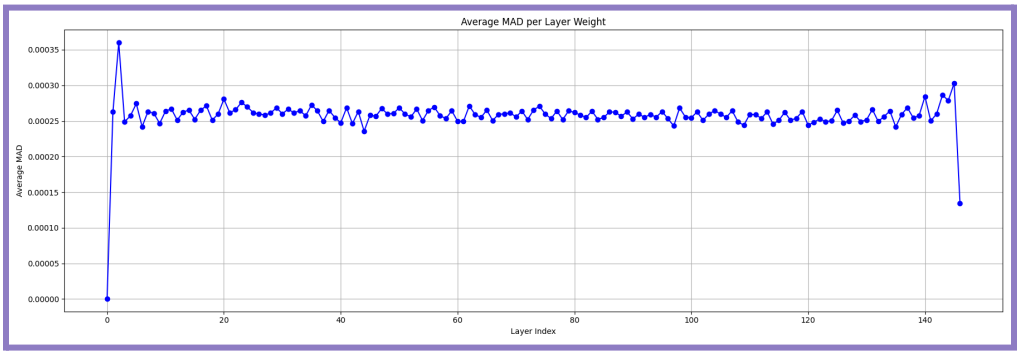


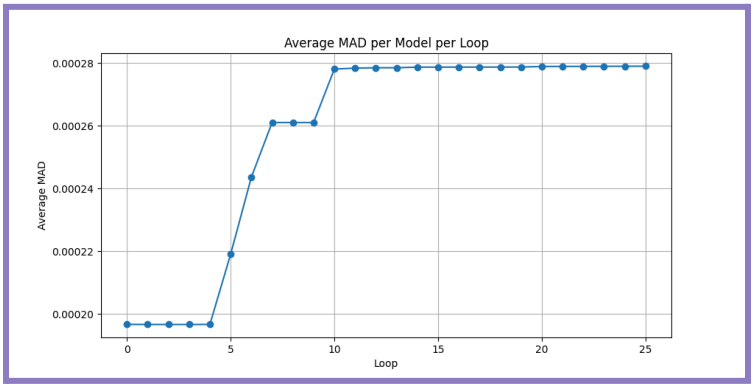
Stupid check even if the weights look similar,
It answers the eval terribly.
But what if I kept training it?
Followup says it is still collapsing but slower

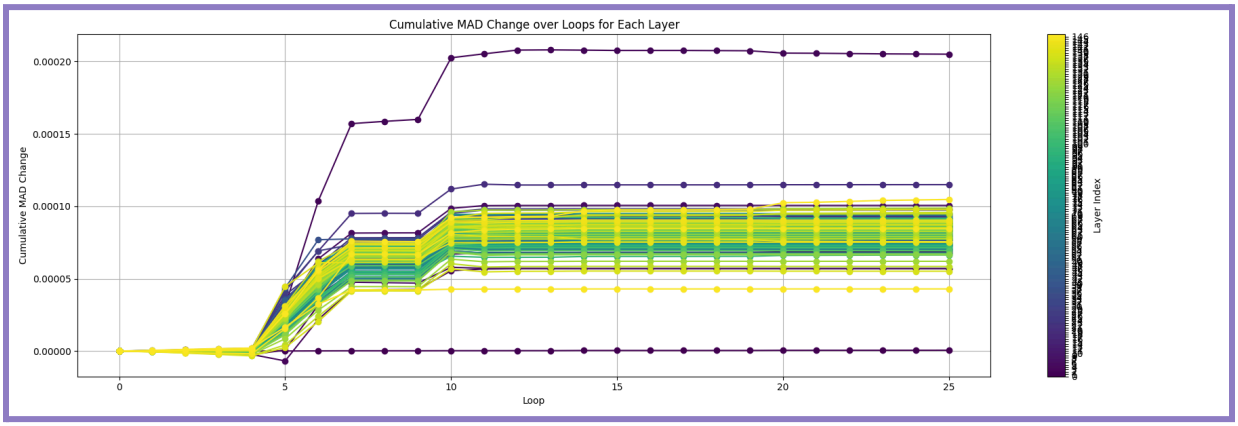
▾ **hellaswag**
alias "hellaswag"
acc,none 0.264389563831906
acc_stderr,none 0.004401063265803201
acc_norm,none 0.2755427205735909
acc_norm_stderr,none 0.004458742356237853
▾ **lambada_openai**
alias "lambada_openai"
perplexity,none 826737930.0258541
perplexity_stderr,none 108614099.67728662
acc,none 0.00019406171162429653
acc_stderr,none 0.00019406171162430181

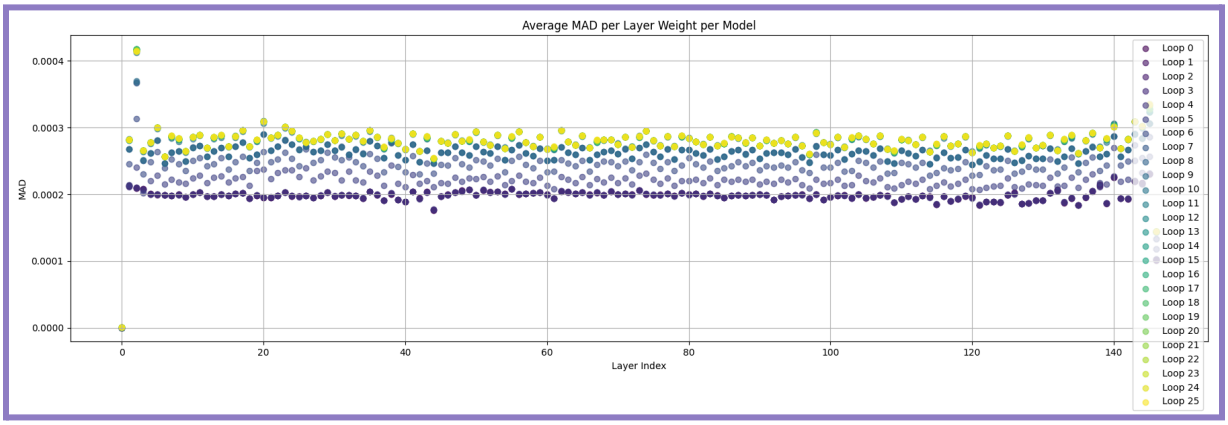
7-11-24: combined analysis of layers, LR, and eval

layer_analysis_50_100_high_learning_5epochs_models

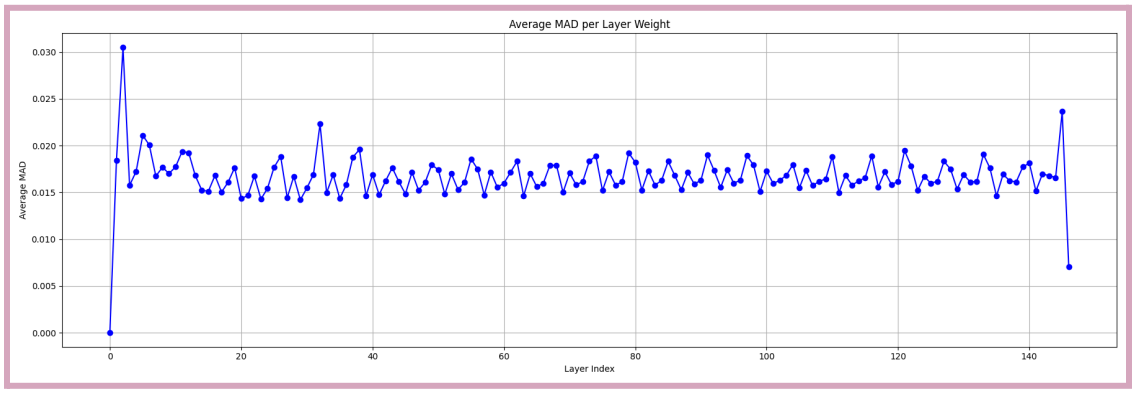


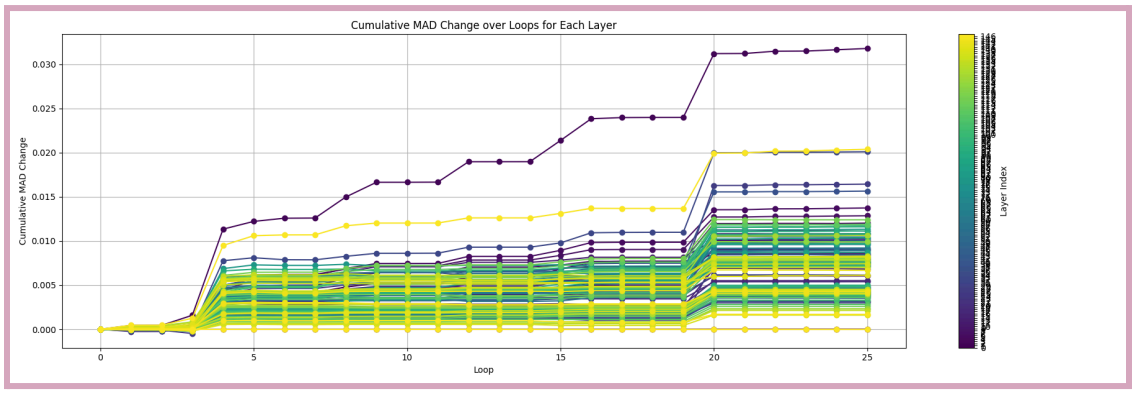


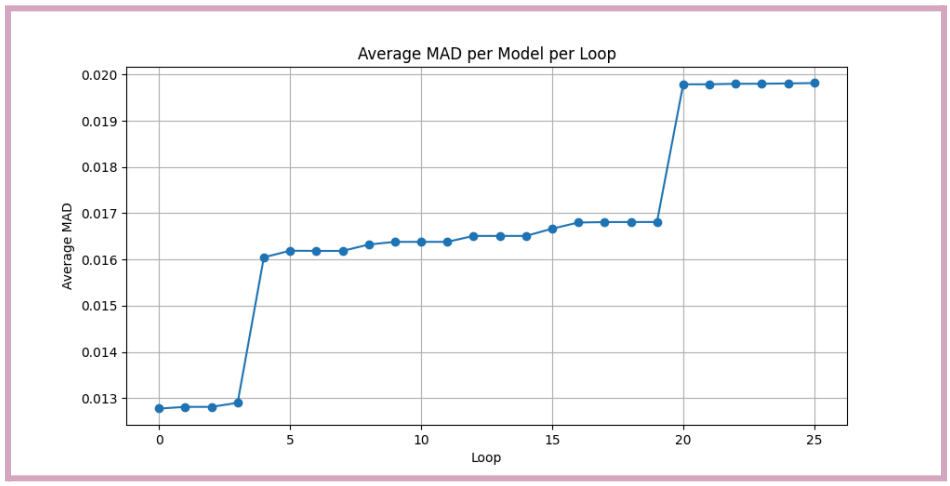


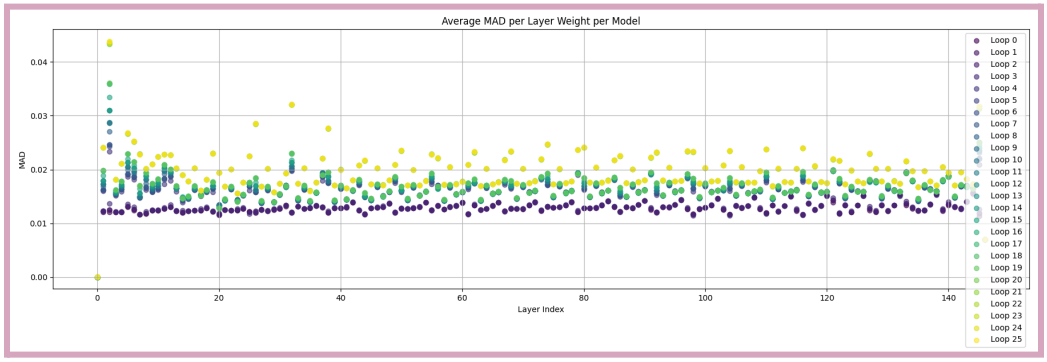


Layer_analysis_0_no_learning_5epochs

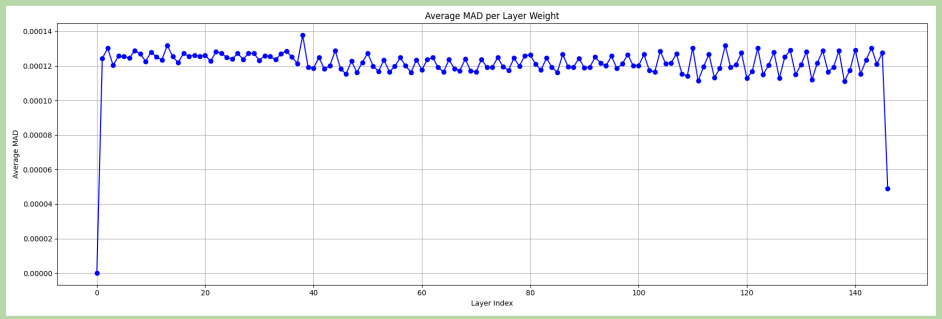


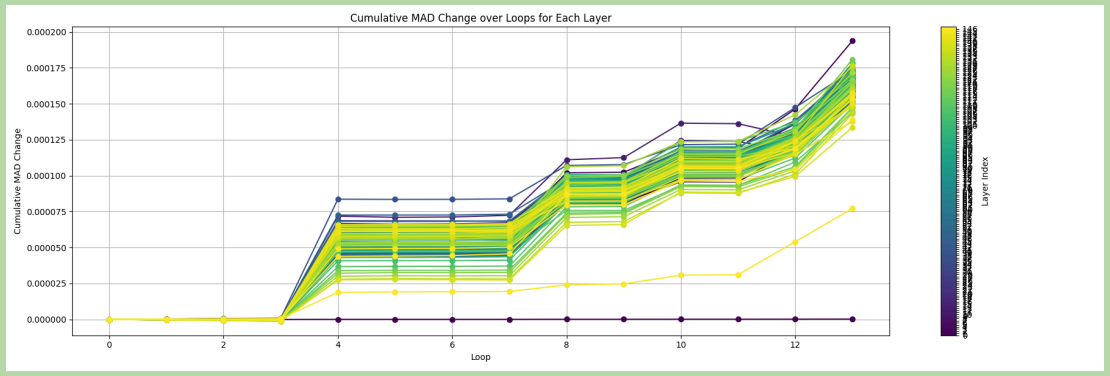


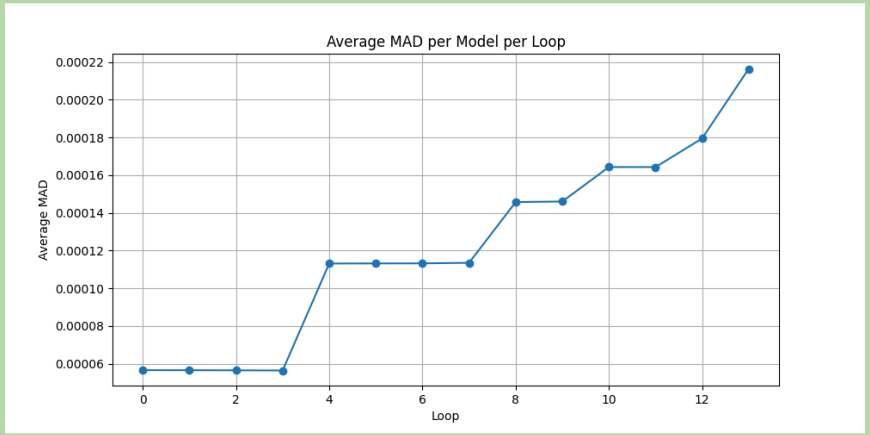


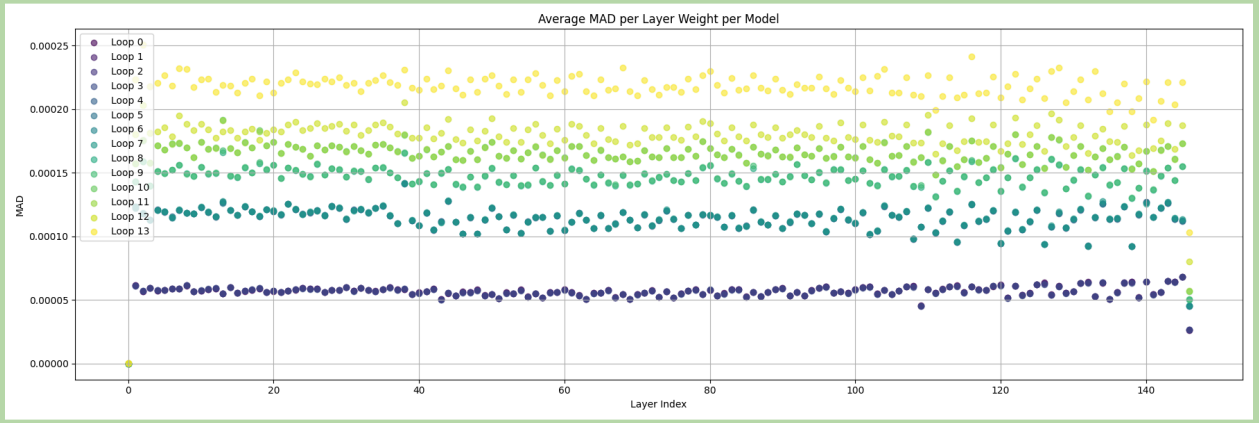


layer_analysis_1_50_medium_learning









Conclusive thoughts from layer study:

High error from being incorrect results in a massively degrading model. Of course the models that select correct settings collapse less because the error in back propagation is so much lower.

Higher number of correct has a significantly lower training loss leading to much lower error propagation.

So basically these results above just show greater error equals more collapse. Important to know but not groundbreaking. It's still very interesting that when training on these you could potentially choose the settings with the least loss per step. This would allow very little model collapse while continuing to learn.

We can mitigate collapse by simply choosing the LR that produces the lowest train loss. This would mean we are likely learning while also damaging the model the least amount.

Main goal in fine tuning a pretrained model is to keep training loss as low as possible per step.

Per training step:
No back prop, find the lowest loss LR combination, then train on that.

Steps:
Search for a LR value by
LR: train, check loss on model,
- search for lowest loss.

7-12-24: evaluating training LR searcher

Even with very loss loss error in training often the model will not get the correct responses.

Does low loss actually lead to less collapse?

The correlation of loss to correct answers is not as clear as once thought.

I also am now unsure if high losses actually lead to model collapse, perhaps my study was not clean enough before and I am not sure how loss is correlated with the MAD values. Yes the losses after training 5 epochs is low on that particular model but I do not know what the losses looked like in between the 5 epochs leading to those correct responses.

There might be a required minimum loss or change to the model per memory formed metric I need to develop to fully understand this method.

Loss is loosely correlated with memory. Certain questions seem to require a level of loss in order to remember that is different that other questions.

After 3 epoch training on same LR the lowest loss does mean it remembers but sometimes a loss rate very similar after 3 epochs remembers nothing.

Even more unusually the loss after training the 3rd epoch is different than the loss at at epoch 3?? Sometimes drastically so.

##
https://discuss.huggingface.co/t/what-does-hugging-face-trainer-do-special/97031
Comparisons of the different trainers:
Hugging Face Trainer Correct Count (Batch): 744
Manual Training Loop Correct Count (Batch): 116
Hugging Face Trainer Correct Count (Single): 9
Manual Training Loop Correct Count (Single): 2

After update:
Gave up, no idea

##

7-15-24: LR and Loss continued

Was wondering

1. Map the loss requirements for a model to learn a question.
    - Does it take a particular amount of loss for a model to remember something?
    - What does the loss graph look like for remembering something?

2. Is model collapse required to remember something? Is there an exact size I can observe after it learns something?
    - I could build a slow loss curve with low learning rate and compare the MAD
        - And run llm-eval
    - What would the MAD look like of a slow loss rate curve versus a fast one?

3. On both of the questions above how do batches work?
    - Is a batch literally the LR curve of all the questions combined into one?
    - MAD of batch vs single

Note: be careful about inference do_sample=True allows a gradient of learning instead of on off

The trouble I had was that loss decrease does result in a model learning the information however the change there is not as expected. This may have been a result of the do sample not being used.

Loss does correlate to correct count however it is elusive because each training epoch is a new LR landscape to search through. So as it trains even if that lr is the lowest the next epoch is not going to be the best LR.

So per epoch perform a LR search. This should be the least damaged and best performing model. The do sample really confused my results.

I am still seeing weirdness around loss decreasing and correct count decreasing.

Train Loss = 9.582703590393066, Inference Loss = 1.4603519439697266, Correct Count = 17, Grad Norm = 153.2225189208
Train Loss = 9.582703590393066, Inference Loss = 2.4400274753570557, Correct Count = 11, Grad Norm = 153.22251892089
Train Loss = 9.582703590393066, Inference Loss = 0.7903410196304321, Correct Count = 10, Grad Norm = 153.222518920898
Train Loss = 9.582703590393066, Inference Loss = 0.47332295775413513, Correct Count = 23, Grad Norm = 153.22251892089
Train Loss = 9.582703590393066, Inference Loss = 0.3606114983558655, Correct Count = 10, Grad Norm = 153.22251892089844
Train Loss = 9.582703590393066, Inference Loss = 0.32670363783836365, Correct Count = 7, Grad Norm = 153.22251892089844
Train Loss = 9.582703590393066, Inference Loss = 0.32362130284309387, Correct Count = 6, Grad Norm = 153.22251892089844

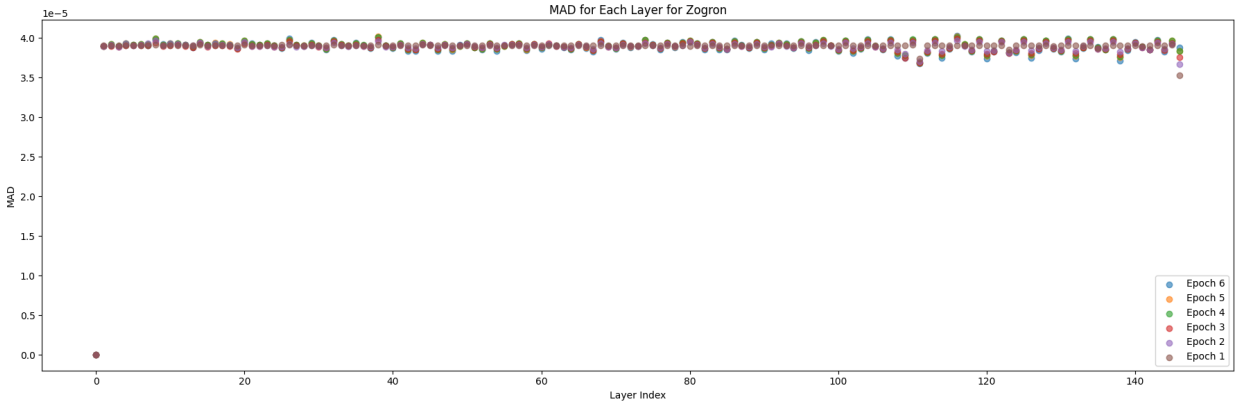
This still may just be noise?

This example annoys me alot:
Epoch: 2
Train Loss = 0.32226496934890747, Inference Loss = 74.5011978149414, Correct Count = 483
Vs the best model of that search:
Train Loss = 0.32226496934890747, Inference Loss = 0.17933180928230, Correct Count = 58
Epoch 3: again, annoying
Train Loss = 0.17933180928230286, Inference Loss = 0.67578428983688, Correct Count = 723
Train Loss = 0.17933180928230286, Inference Loss = 0.1336868256330, Correct Count = 121
Epoch 4: best choice:
Train Loss = 0.133686825633049, Inference Loss = 0.07525704056024, Correct Count = 577
Epoch 5: best choice
Train Loss = 0.07525704056024551, Inference Loss = 0.0701844096183, Correct Count = 608

Could this be implying it was a large update to the early layers? Allowing easy memory but very damaging? Seems like a nice gradual learning of the info.

What is happening to the model's weights and eval score when it is doing this low to high loss change? But at the same time it is getting the info correct more often?

A new best in model collapse reduction. This is the lowest value I have had yet.

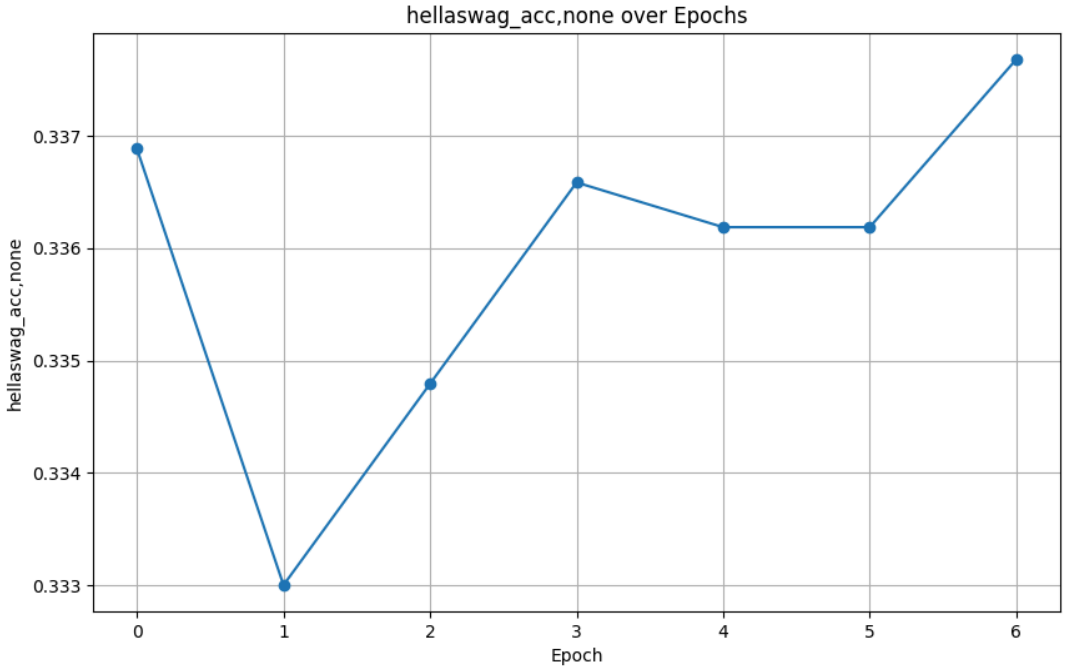


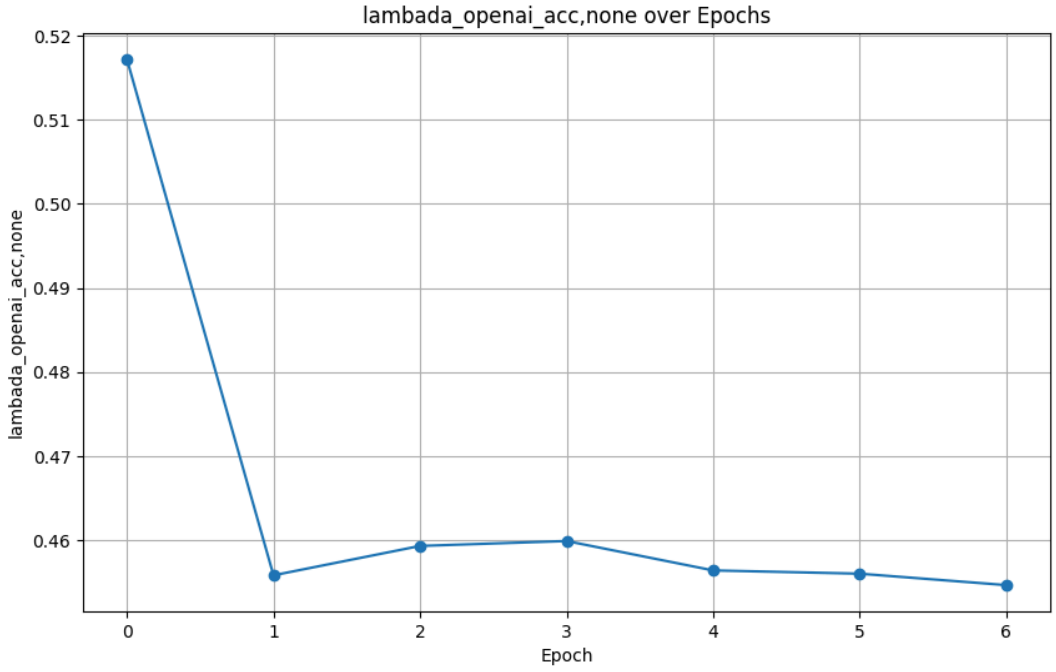
Interestingly this is closest to the result of "medium" learning and not only that but I also find that the highest correct count was not the lowest LR so in that previous study.

- Partially confirmed, looking over medium learn many of the "medium" choice values were very low training loss. Sometimes much better, example below: 1_50_medium/2/results.csv

Train loss, correct count
0.44, 99
0.14, 45
Seems I accidentally chose the lowest loss value when looking at medium ones. But since I am particularly choosing the lowest learning rate in this method it is better in model collapse metrics.
Method is also very reproducible. I get very similar results on my replica. Very reaffirming to my other idea that loss landscape changes per epoch.

Zorgon Question llm-eval results. Epoch 0 is base 410m model.





Overall, incredible results. I am curious about why the lambada metrics collapsed so much but never recovered unlike hellaswag. I need more data to confirm these results, can it be replicated on a batch?

Conclusion and thoughts 7-15-24:

- I want to do the exact same test above with a batch of all the examples. I wonder what it will look like.
- I need to confirm the results one last time by doing the exact same training for the exact amount of time and compare the collapse.
    - Or have I already in previous tests?
    - Go look over the previous study choosing highest correct value

- If batch training performs nearly the same way this is ready for prime time.
- Is it possible that the initial training loss for each data input IS the way I could map it's plane out? Is that a way of measuring the required training per datapoint?
- Build a LR prediction model?
- **What's a shuffled dataset do to the LR loss plot for a batch?**
    - **Does every shuffle result in a new LR/loss plot?**

Does a smaller LLM trained the exact same way (pythia) allow me to do the LR search on the smaller model? Allowing massive speedup in the search?

I should compare single LR training until it reaches a level of loss.

Paper Validation:
Train on multiple different LR's until they reach a training loss.
Train a few models in LR searcher method, shuffled dataset.

Compare the model collapse metrics and evaluation scores after training across the epochs.

What should my stopping metric be? It seems like slop of change is the only metric I could use.

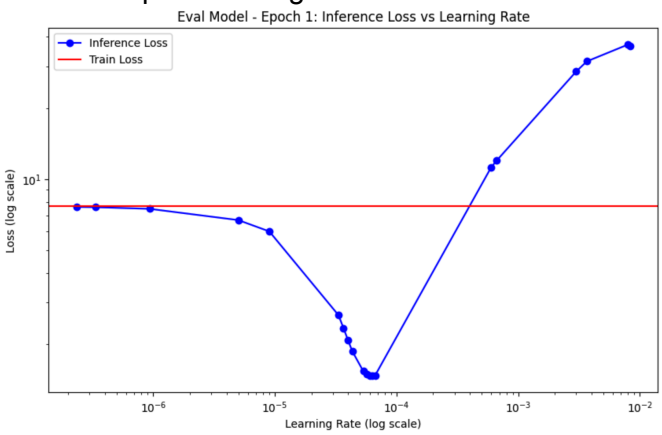
7-16-24: batch effects lr searcher

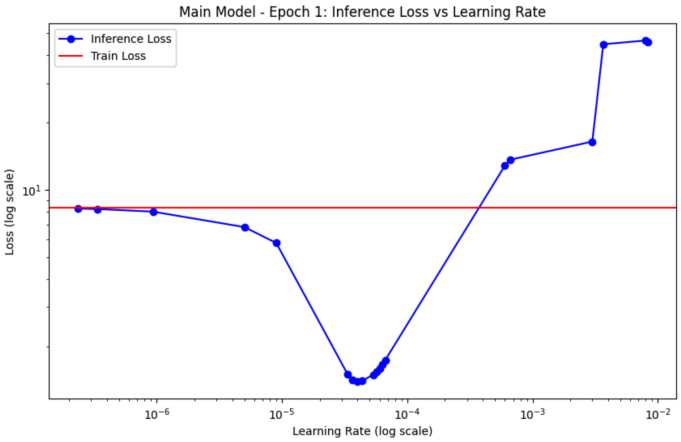
+ smaller model comparison

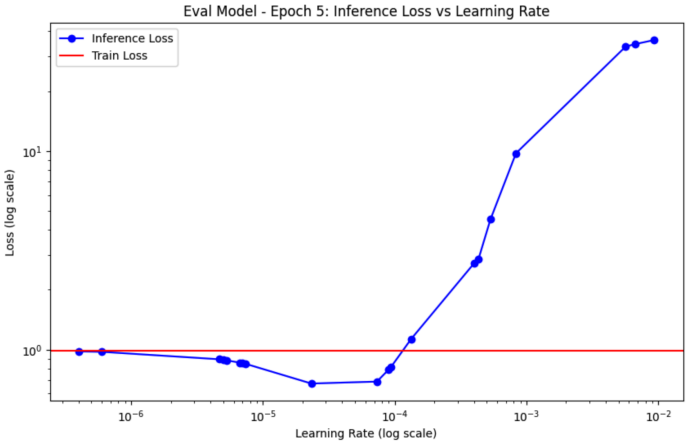
Still see very specific phenotype of model learning by large increase in loss:
Train Loss = 0.3314768970012665, Inference Loss = 49.295162, Average Correct Count = 28.3

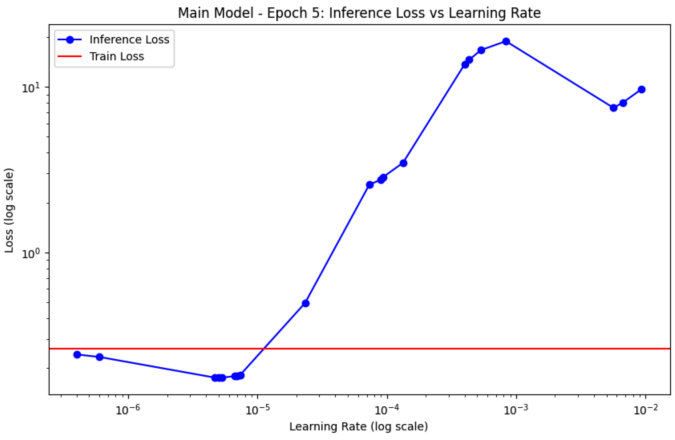
Early eval model comparison shows it is tantalizingly close. Not 1:1 it seems scaled and occasionally has different results.

VERY similar results. May be able to use it if I assume error. I want to watch it for a longer period though before I am certain.





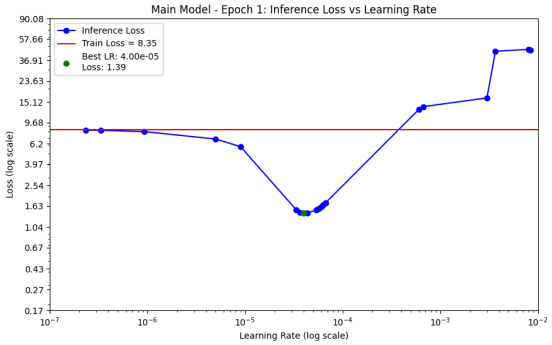


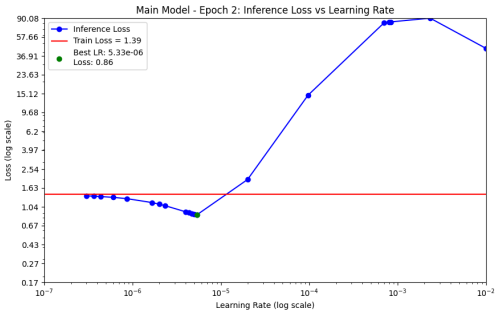


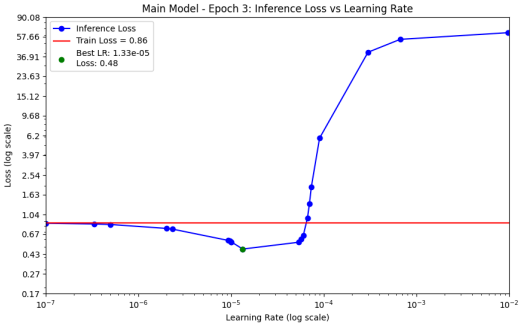
They are very similar but clearly diverge over time. Checking with a higher number of epochs as we get closer to more correct answers.

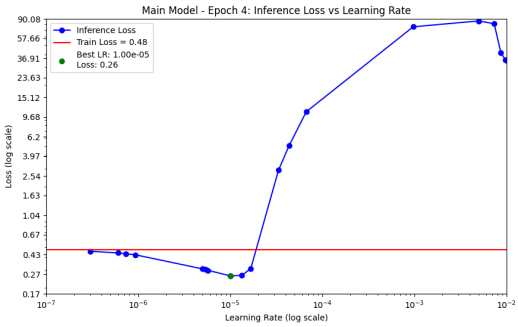
If they diverge so much, how come the first epochs are so similar?

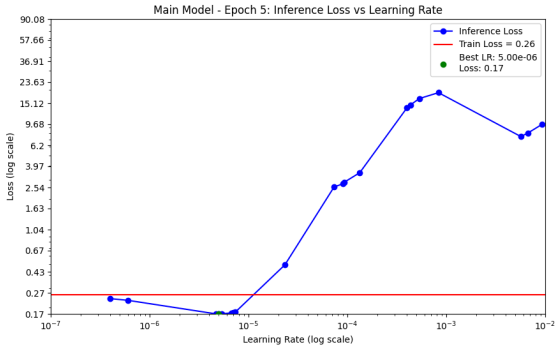
Nice plots on test1 of just main model:





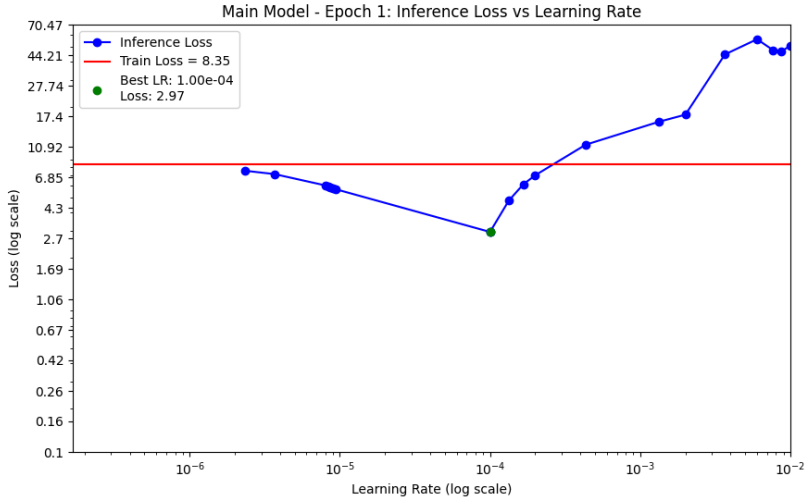


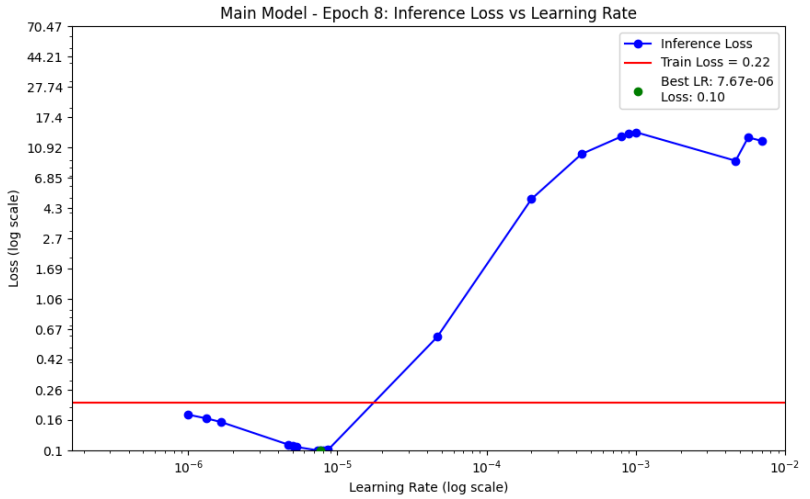


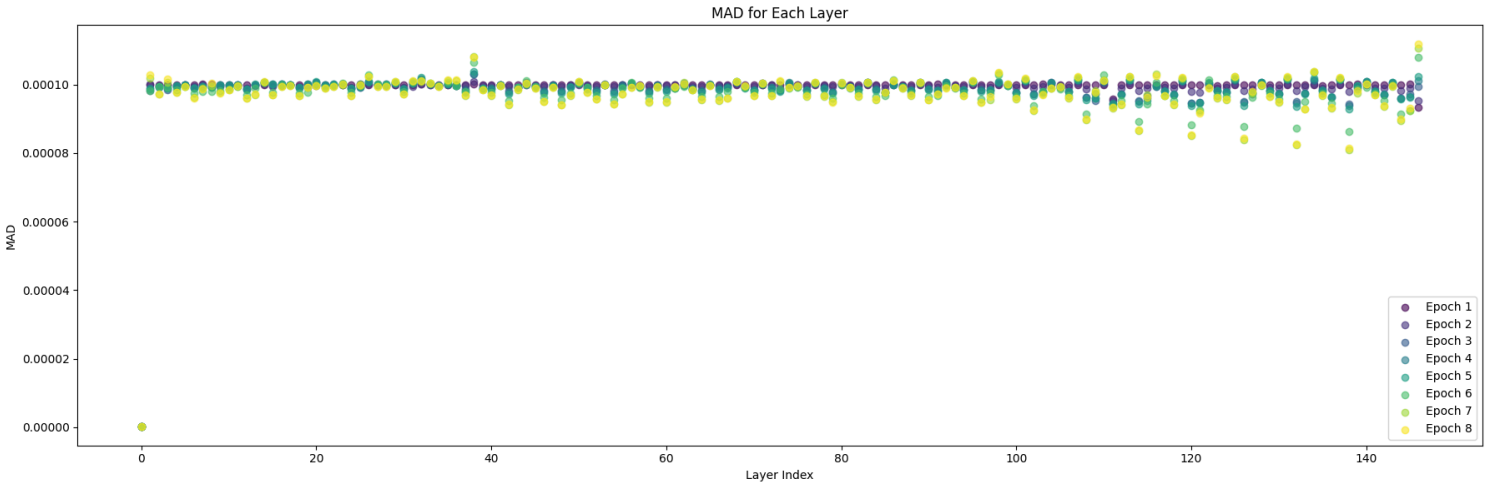


Very nice plots confirming a changing LR loss landscape. Results are confirmed, need to perform collapse analysis.

Test 2, 8 epochs:







Mad plot shows an exponentially worse result than previous single data insertion?

Conclusions 7-16-24

- comparison models are similar but not close enough, maybe used as a first pass but might have too much error for actual LR search

- Seeing LRs change per epoch from the model is very interesting and I should make the same plots but with different methods (such as fixed LR)

- Model collapse metrics show that batches are much worse? My theory that the LR loss landscape is the combined one of the individual values might be true
- Can I confirm this? Run the same test per question on a single step, combine their LR loss values into a single plot?

LR searcher is pretty optimized and I no longer see the need for the fine tuning step.

- Can I use the previous LR plot to predict and search a more accurate area? Less steps but there is a risk that this changes massively after hundreds of epochs.

1. First confirm this actually is better than a fixed LR
2. Then need to start adding in real data and synthetic data to this system.

7-17-24:
LR searcher should not be calculating gradients every time. Calculate once then find the best LR.

V2 times: ~2secs or ~8secs
V3 times: ~0.6secs or ~6secs

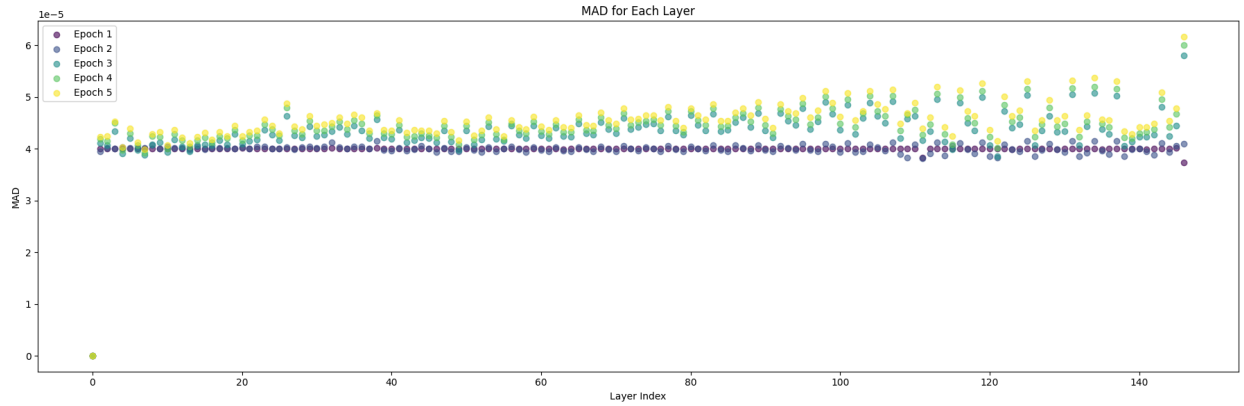
I don't want to fight endless bugs. Just using v2 method.

Fixed LR 1e-5 vs LR searcher:

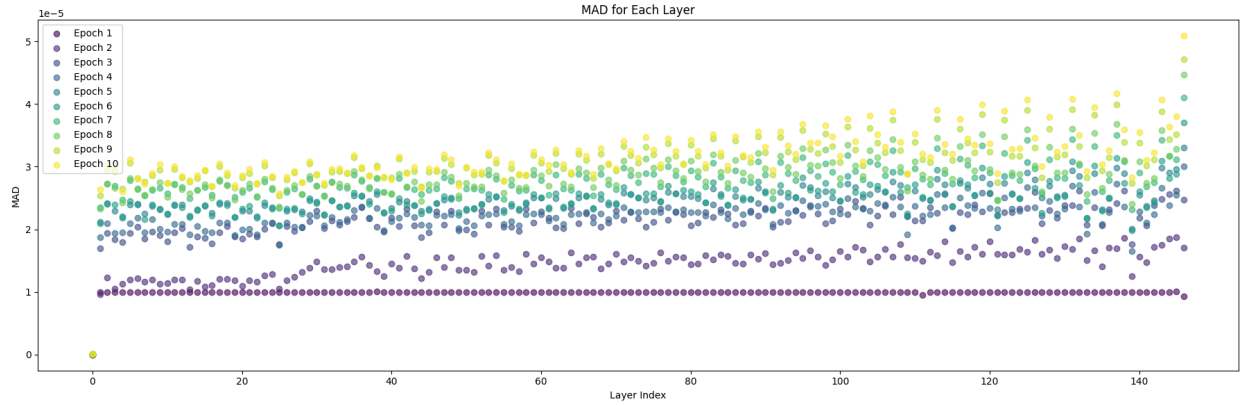
Fixed: 10 epochs: final loss = 0.11
Search: 5 epochs: final loss = 0.10

Notes:
- the companion model seems to diverge heavily from the original with a much flatter learning curve.
- eval model had a higher loss at the end of 10 epochs vs 5

Mad results for the LR searcher show higher differences? But lower change per epoch.
Searcher:



Fixed:



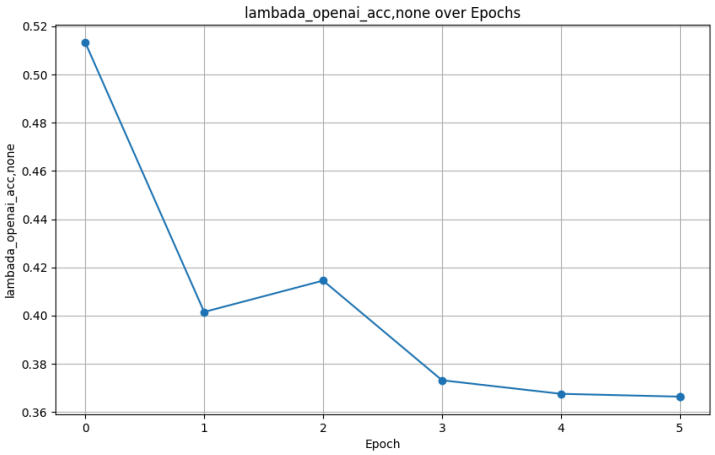
There's clearly a much faster rate of change in the fixed model, however the absolute value is lower than the searching one. Perhaps the 1st epoch from the searcher did poorly? Bug?

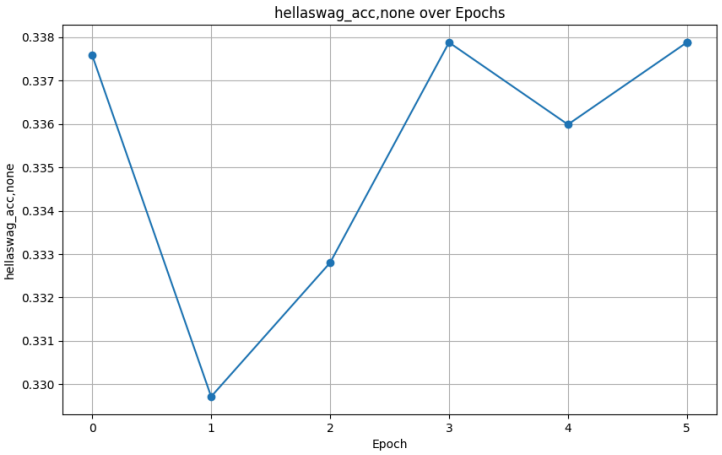
It trained faster with lower LR at almost every step but collapsed more? Will rerun and do an llm-eval
Rerun shows the same results. Lr searcher has ~4e-5 values but lower loss 5 epochs

Learning quicker != correlate with collapsing less

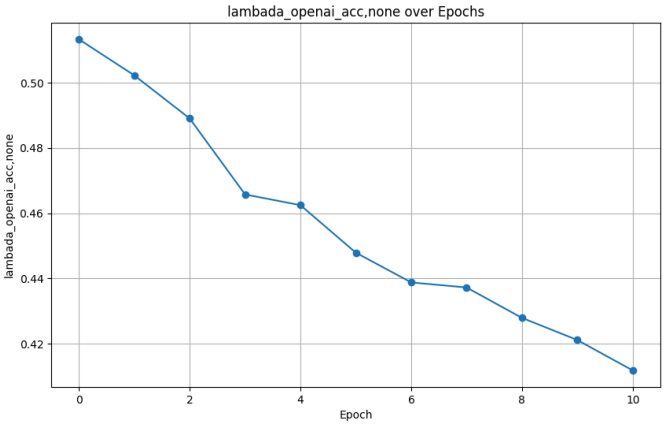
LLM Eval

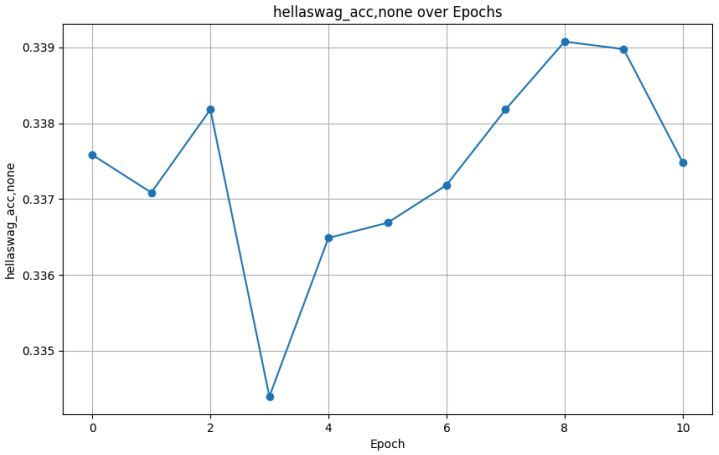
Searcher Batch: hellaswag similar results to single lambada is worse





Fixed 1e5:



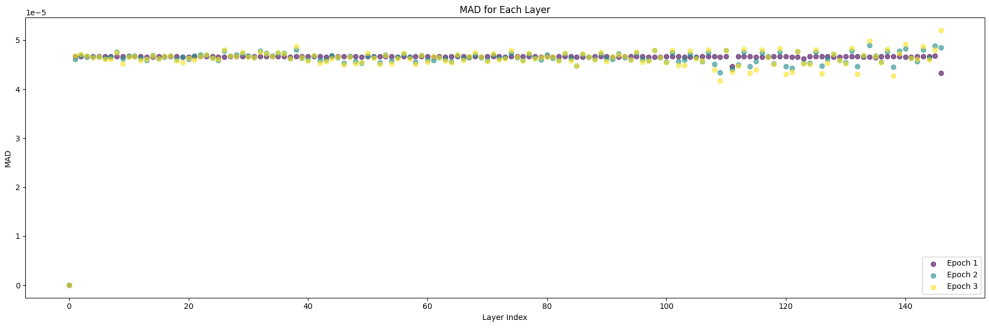


MAD metrics do directly correlate with eval results.

The initial high change in mad from the lr searcher caused the lr searcher to perform poorly. Lr searching for lowest lr loss does not mean lower collapsing values.

LR searching does seem to prevent large changes in the next step but the initial change damaged it the most.

Result is replicated

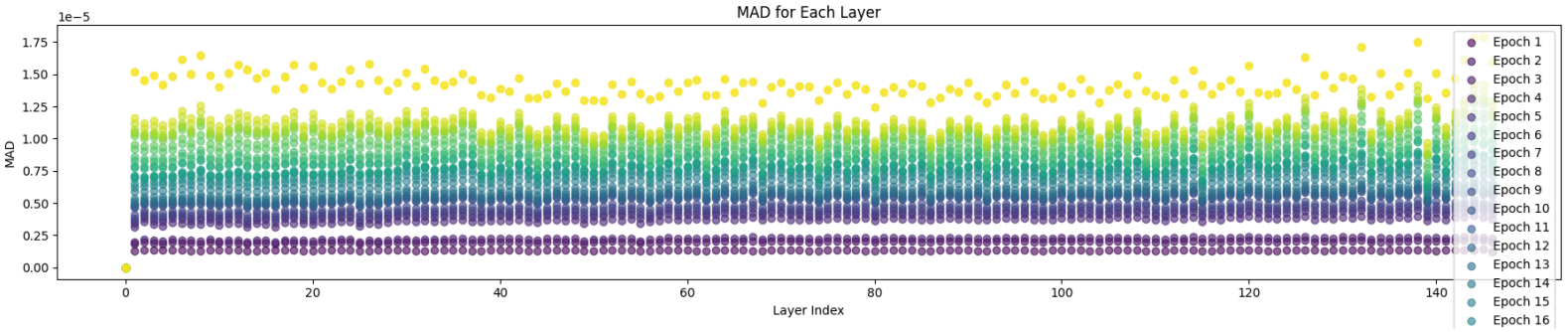


Choosing the lowest LR that is also has a loss less than the train loss.

Is there a lower learning rate settings. Say I find lower learning rate massively decreases model collapse metric via mad.

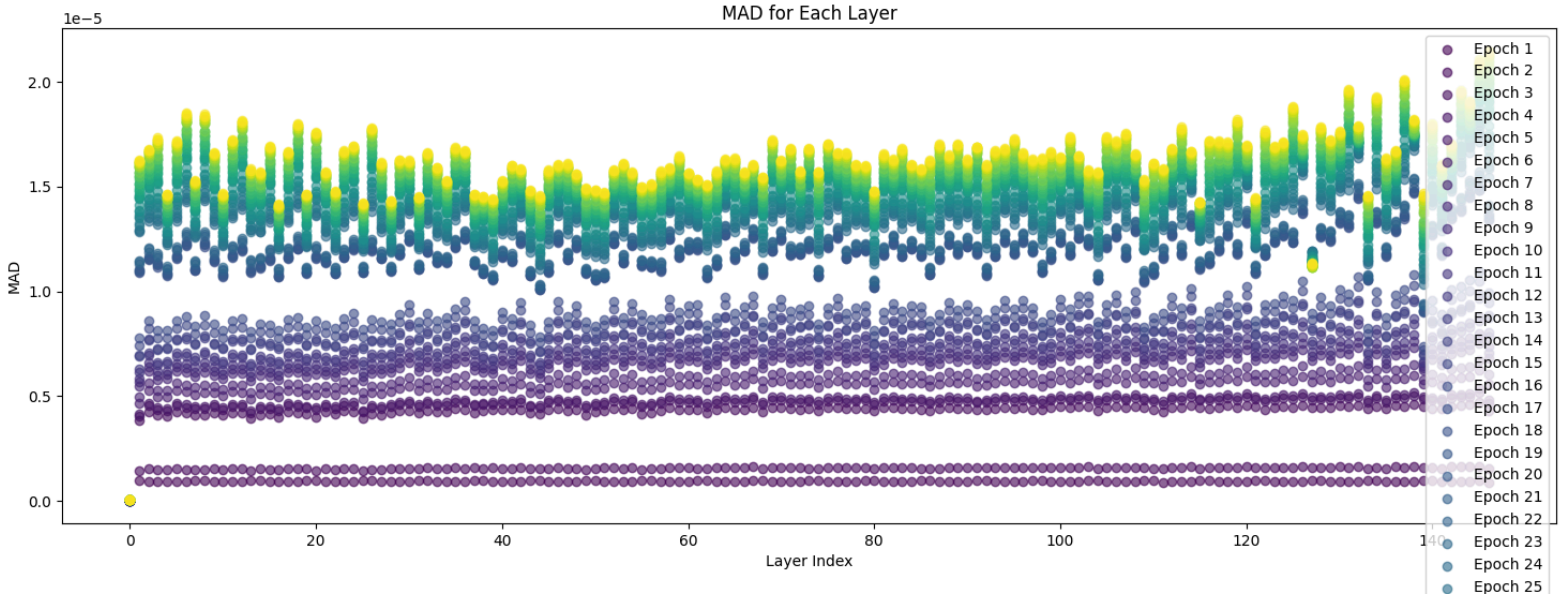
Is there a rate even lower that actually increase MAD? While also improving score? Implying that at a specifically low learning rate (dependent on model size) it genlty enough puts this information in the model and improves it overall?

Single question lowest value. Significantly lower collapse metrics



So lower learning rate does clearly cause less collapse for the same results.
Current theory:
There is a balance of low learning rate and mixed data ratio that leads to CL ability. The faster you want to learn (high LR) the more data mixing ratio you need to accomplish it.

Batch lowest LR:



Lower loss rate decent means less model collapse because lower LR = less collapse

I assume these observations above continue to almost zero collapse ever. If I put loss decent per step to < 1e-5 I'm sure I could insert new info with no model collapse.

But that's slow!

I need to build scaling laws around based on the decent of the LR how much external data do I need to stave off collapse?

Going to a new doc, CL scaling laws.

PLEASE GO GET THE LLM-EVAL of the final model in the latest tests above.

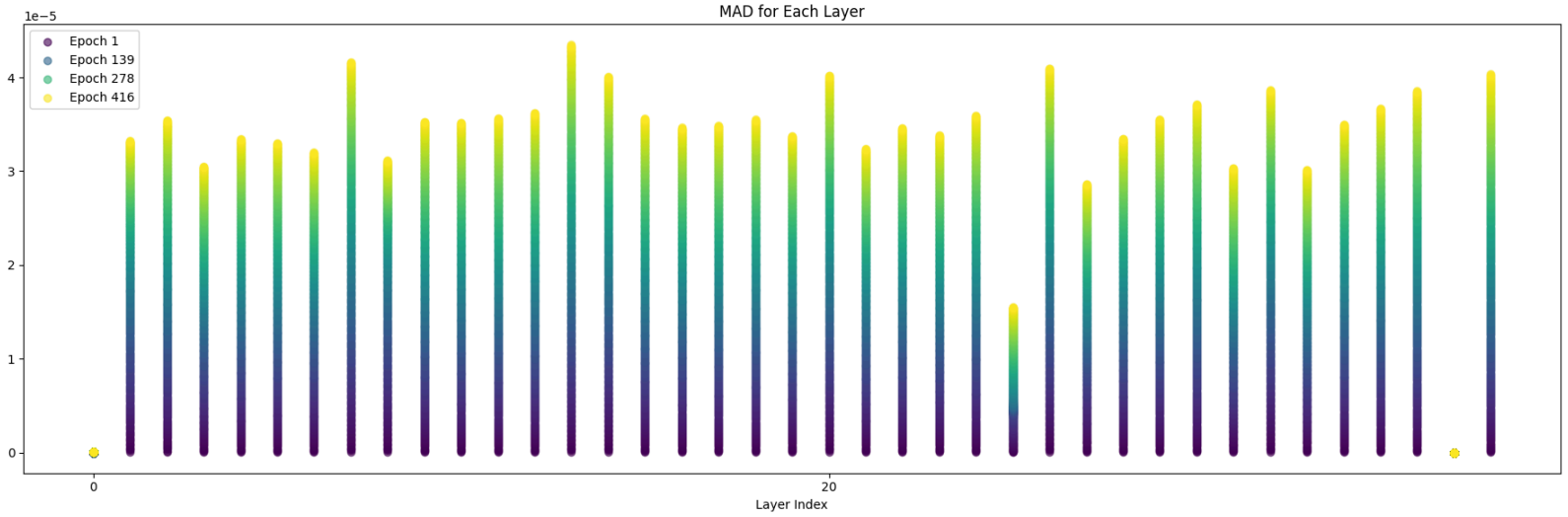
Jiggle Experiment:

- Crazy idea, what if you could calculate the gradient a single time and that is used to train a model fully by simply manipulating the gradient values or learning rate constantly applying learning rate to randomized values.
- Calculate a very low learning rate value, apply these many times.

  How different is a gradient anyhow if your learning rate is very low and loss is very tiny? I bet you it's really similar. Could applying it to a very small randomization of the gradients or lr allow it to learn?

7-20-24:

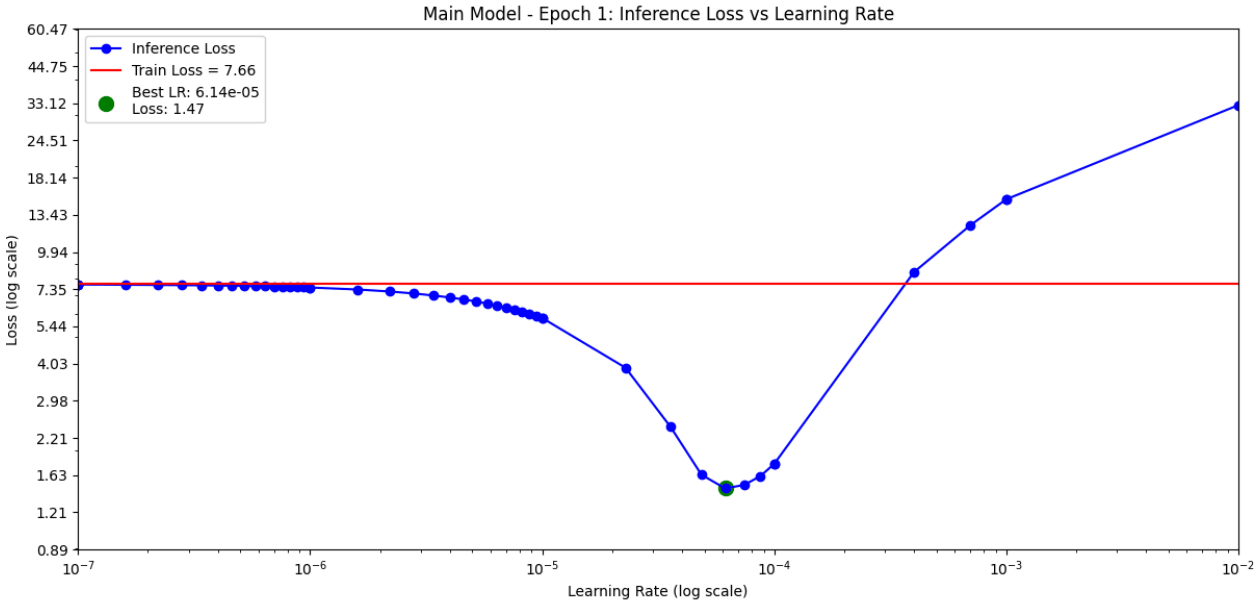
From an initial look at the data below the jiggle idea will most likely not work since an extremely low LR constantly does not save a model from collapse.



So there is a limit on the size of LR and how much collapse there will be. It isn't just to get an infinitely small LR and train forever.

There is clearly a set of values to follow that lower collapse probably the lower LR in the beginning followed by the highest one.

This graph above is from a learning rate of 1e-7 for a batch and hundreds of epochs. It still has not reached high enough accuracy to learn the information.



Other interesting things to note is how the loss of higher values increases and then decreases.

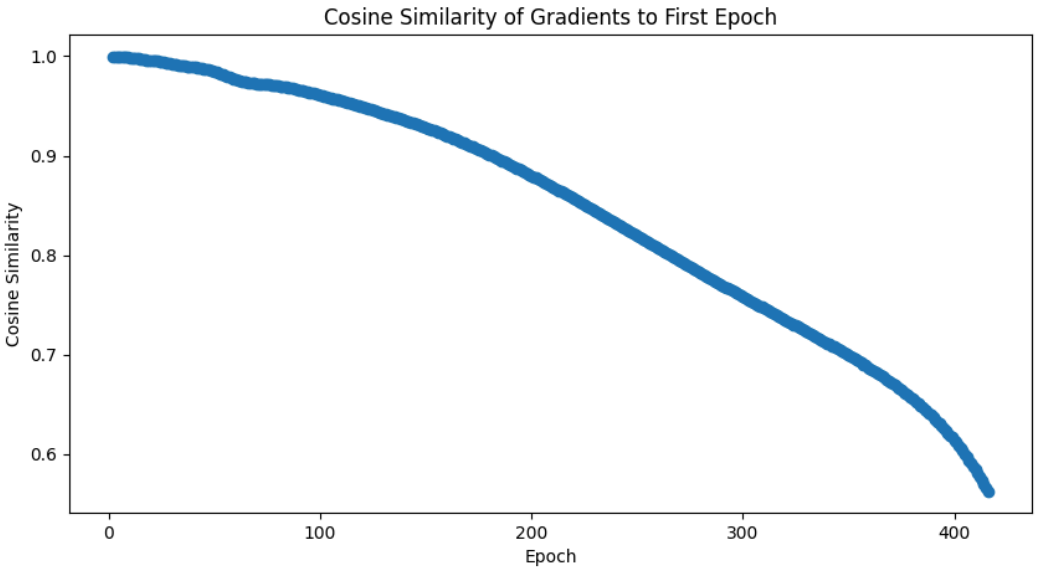
From the previous above mad plots it seems choosing values before the plateau of loss decrease is the ideal location to train and prevent model collapse.

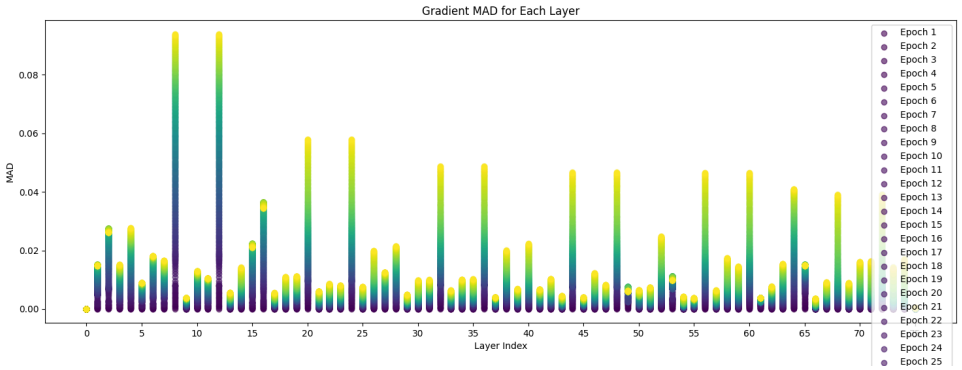
I am still interested in how previous work of finding the best loss decrease value had a higher initial jump in collapse but the following updates where far smaller.

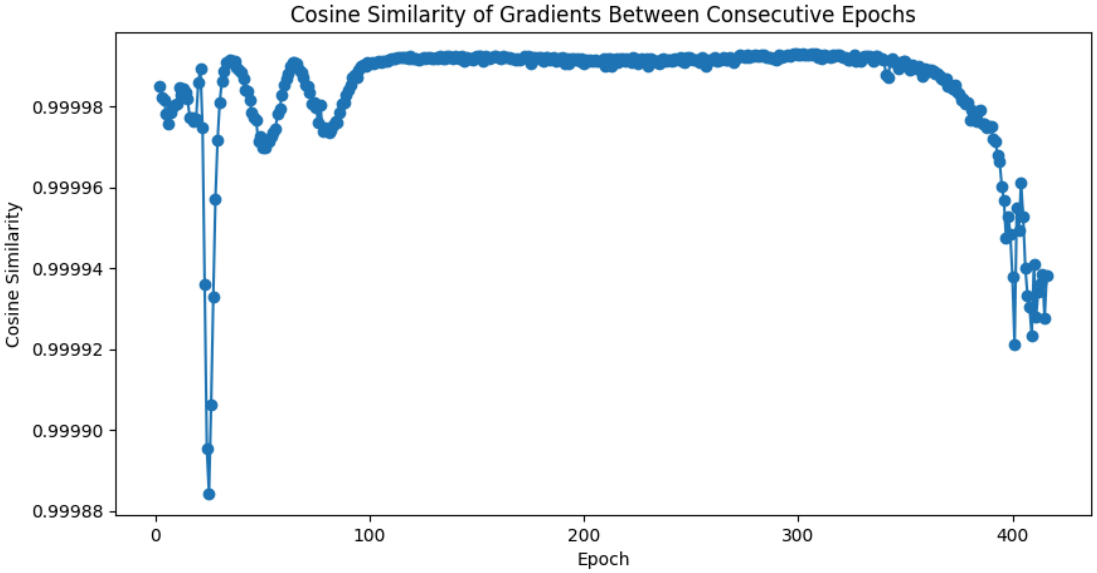
Seems like there is an interesting balance of lowest LR and lowest Loss to be done.

Note that around epoch 350 when the loss values decreased in the higher values the cosine similarity between epochs also decreased by a large amount.

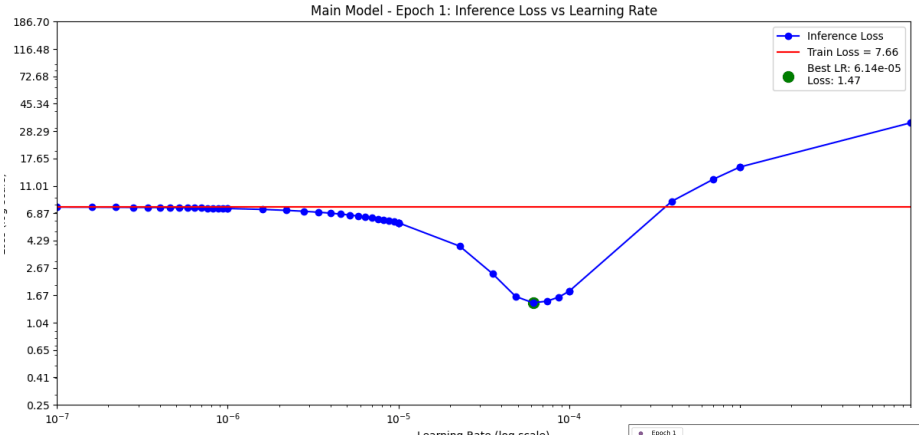
Gradient analysis of low learning rate training.

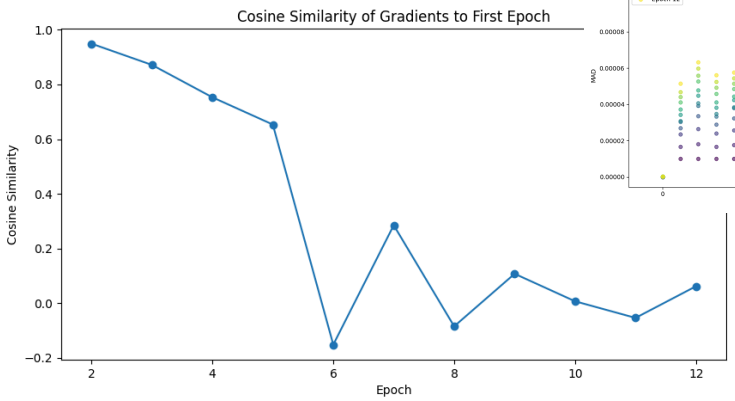


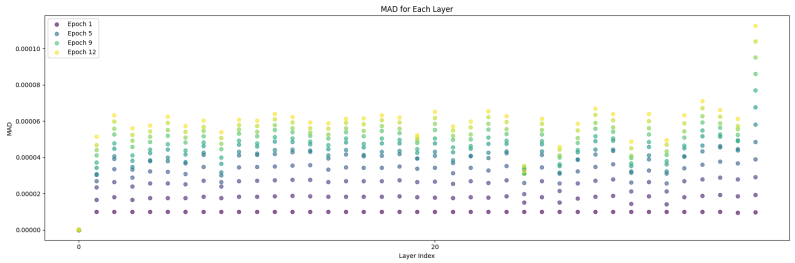


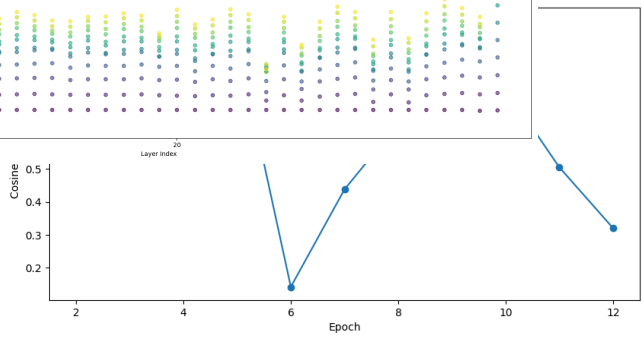


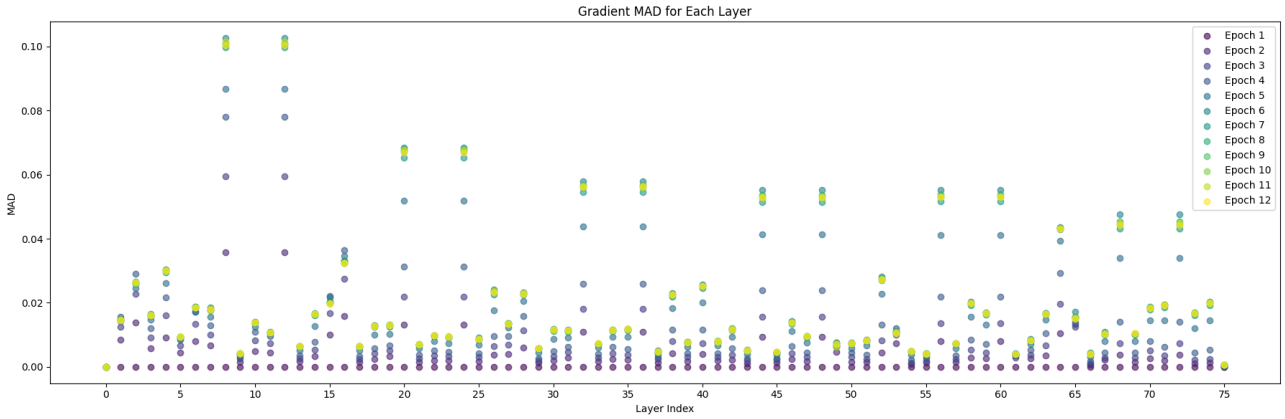
Did the same but for LR = 1e-5









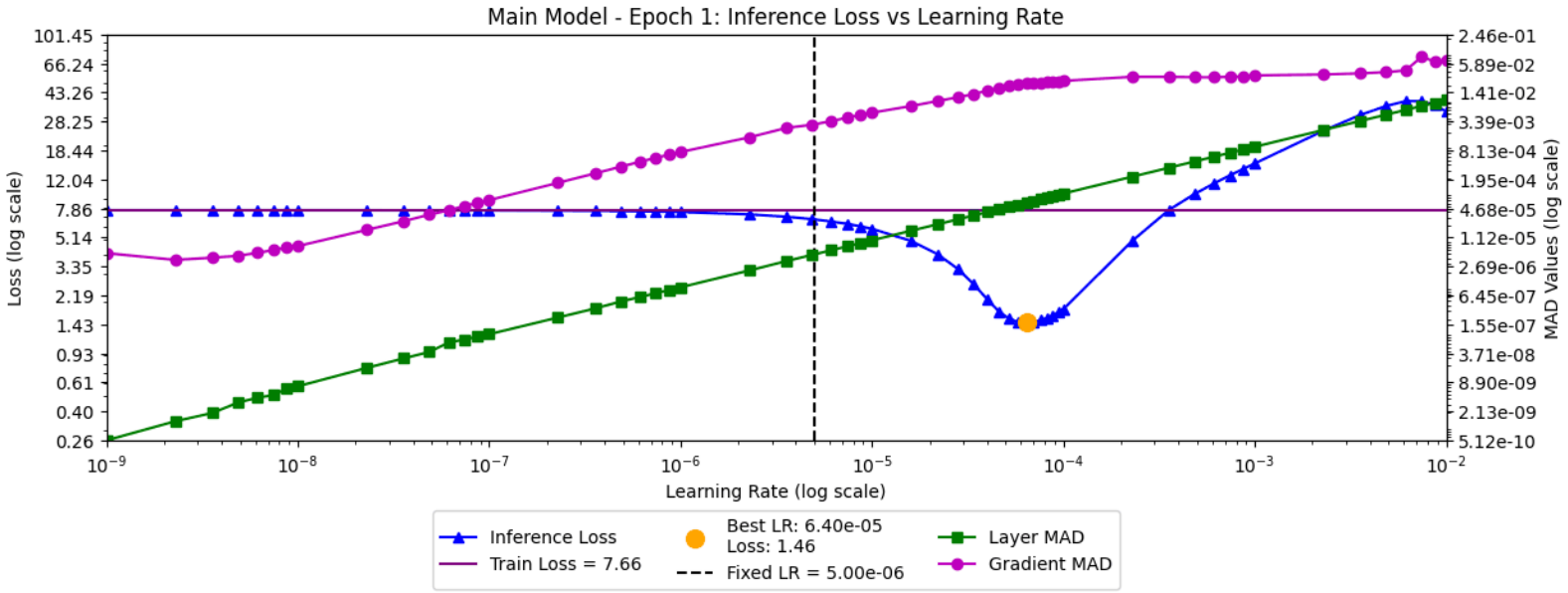


Massive difference in the gradient similarities in the higher LR.

Notice in the mad for this fixed LR that the change to the layer weights is not a linear progression with a fixed learning rate. There is clearly a slowing factor as loss decreases and the change decreases. I will perform a significantly higher fidelity study of gradients and layer mad for every single learning rate to learn its nature.

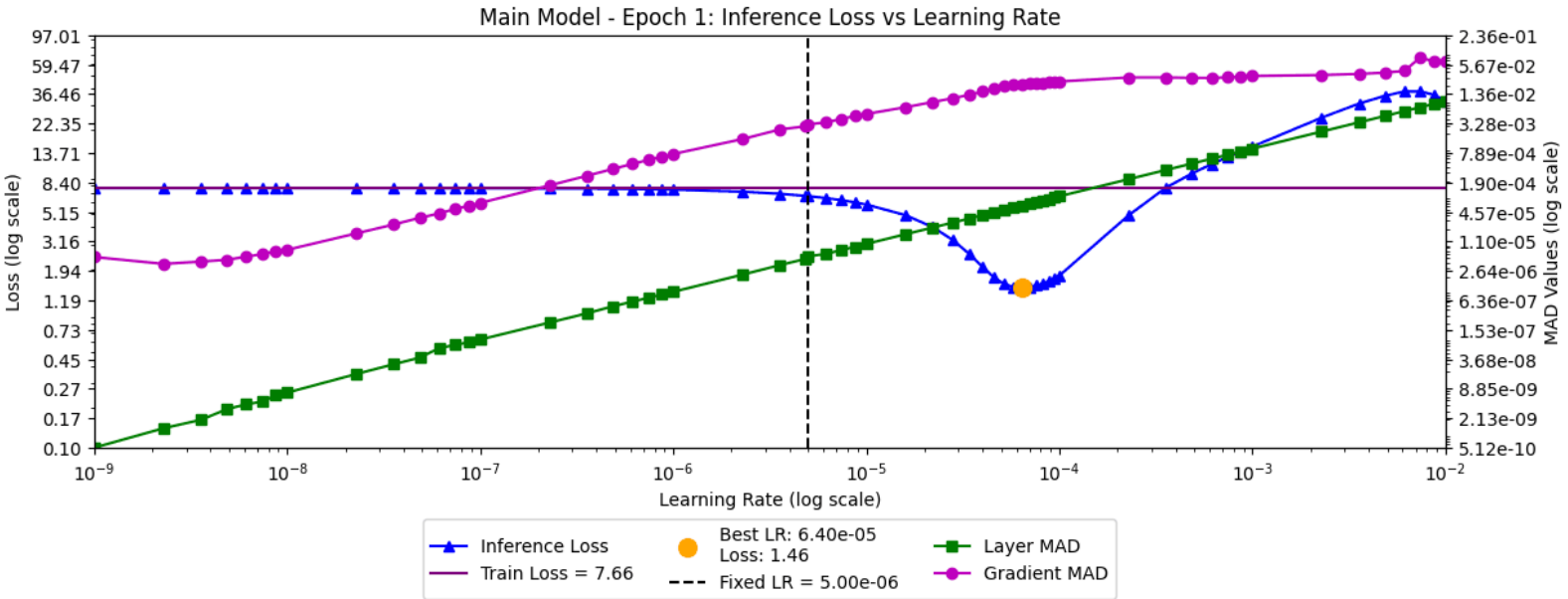
7-19-24: high fidelity mad study 1e-6

Layer collapse metric is not linear but basically plateaus. This means early on I should do a low learning rate and then follow up with the highest learning rate possible before mad starts increasing again. These results correlate with previous mad layer analysis per epoch. Later epochs seem to increase in smaller steps than earlier. This was even seen in the LR searcher which chose different learning rates at every step, unknown why this occurs.

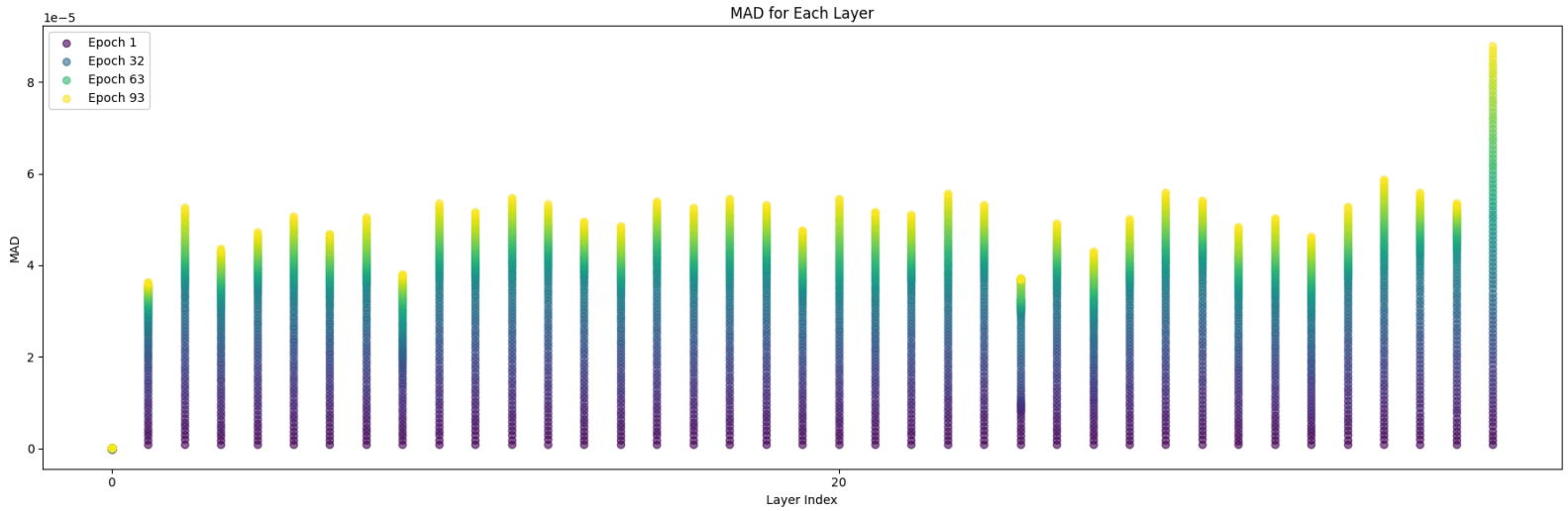


7-30-24: same as above but 5e-6

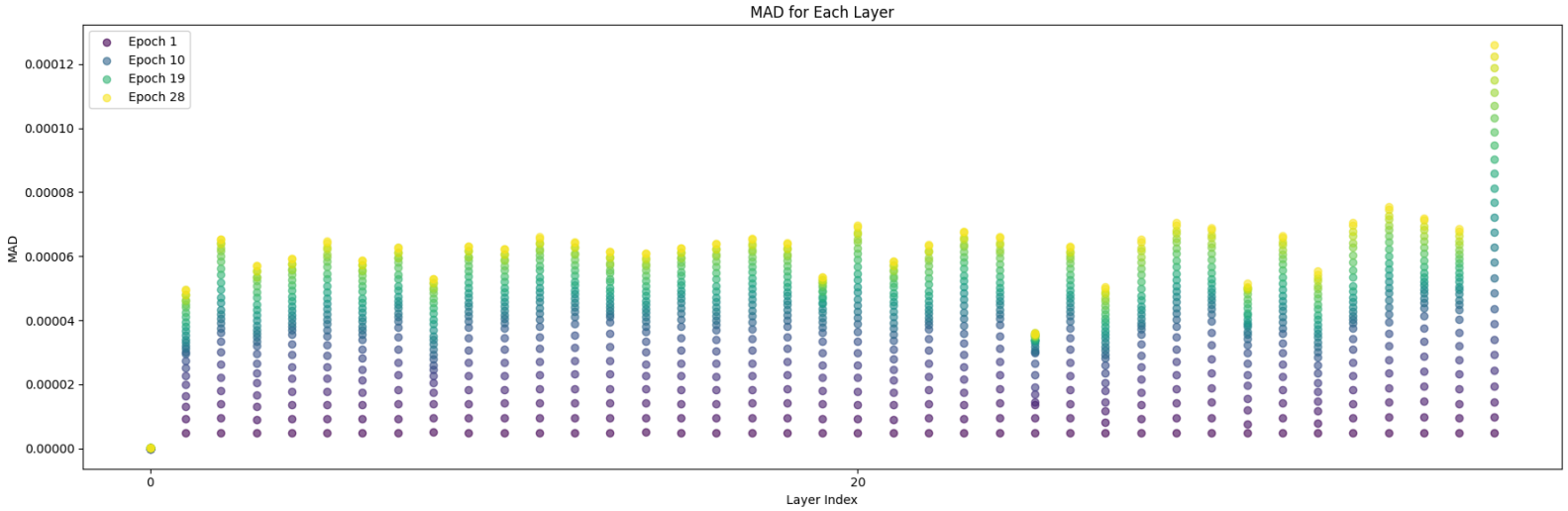
Note: It seems this work is replicated in other papers. An identical effect just seen in a different way. They find a warmup followed by decay best supports CL objectives. Which is exactly what this result shows. Reassuring I am correct in my study.
https://arxiv.org/abs/2403.08763
https://arxiv.org/abs/2107.05855



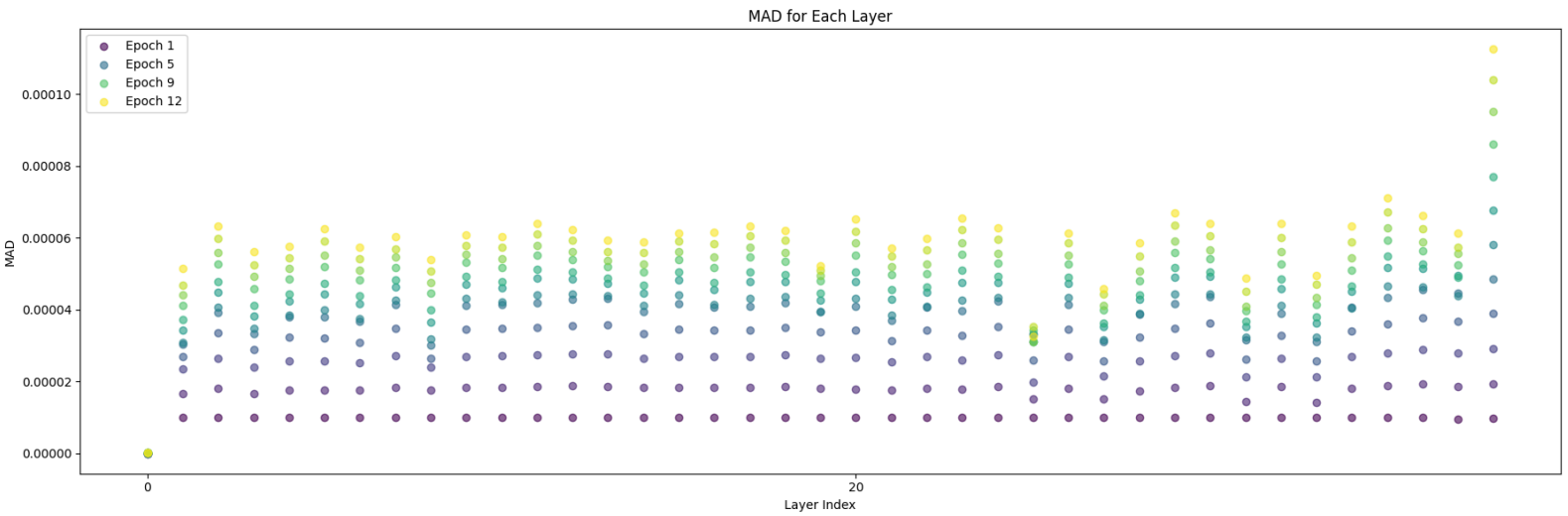
Fixed 1e6 mad plot



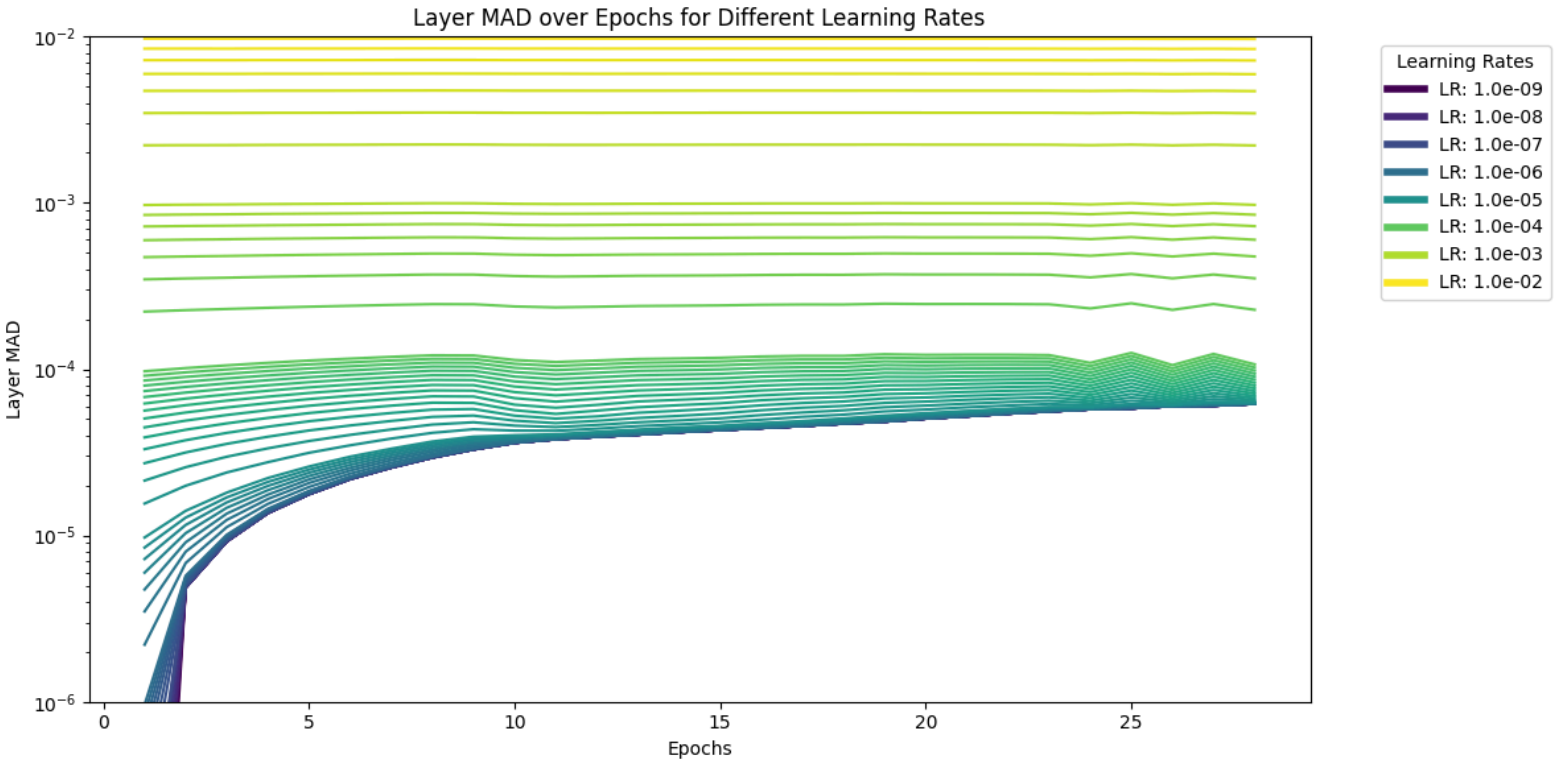
Fixed 5e6 mad plot



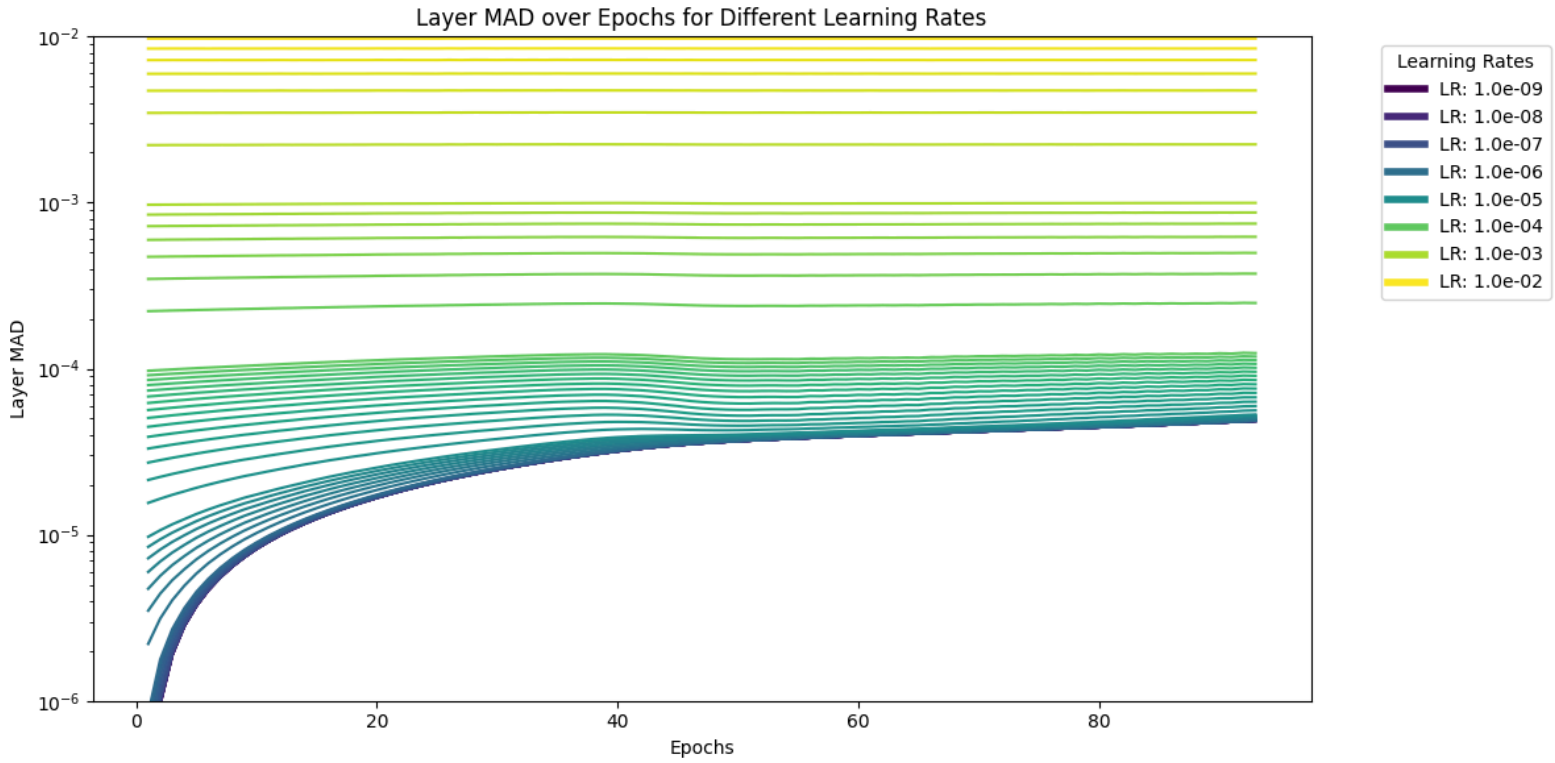
Fixed 1e5 mad plot



Fixed 5e-6

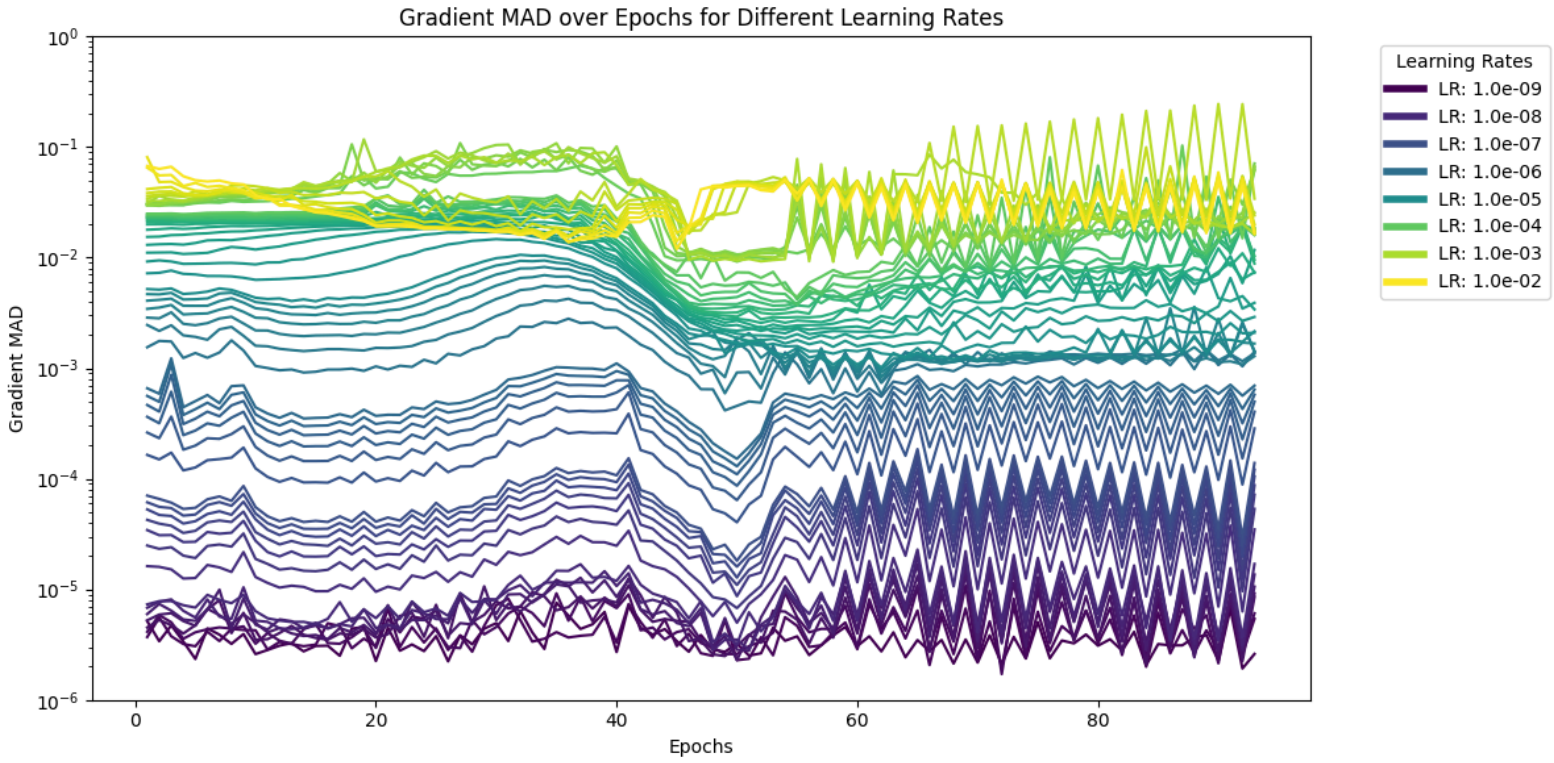


Fixed 1e-6

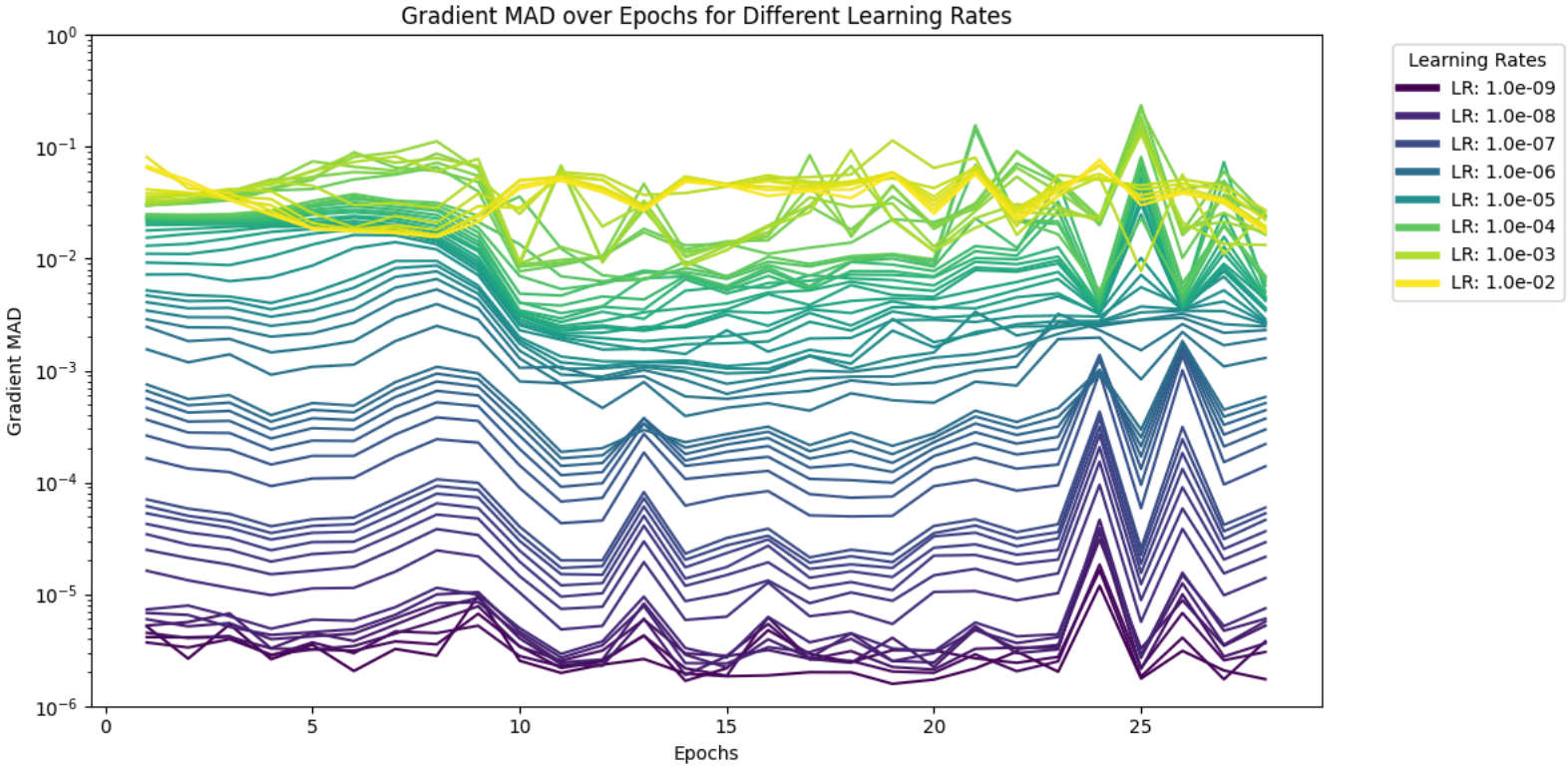


Not even sure what these plots imply

Fixed 1e6



Fixed 5e6



Notes.

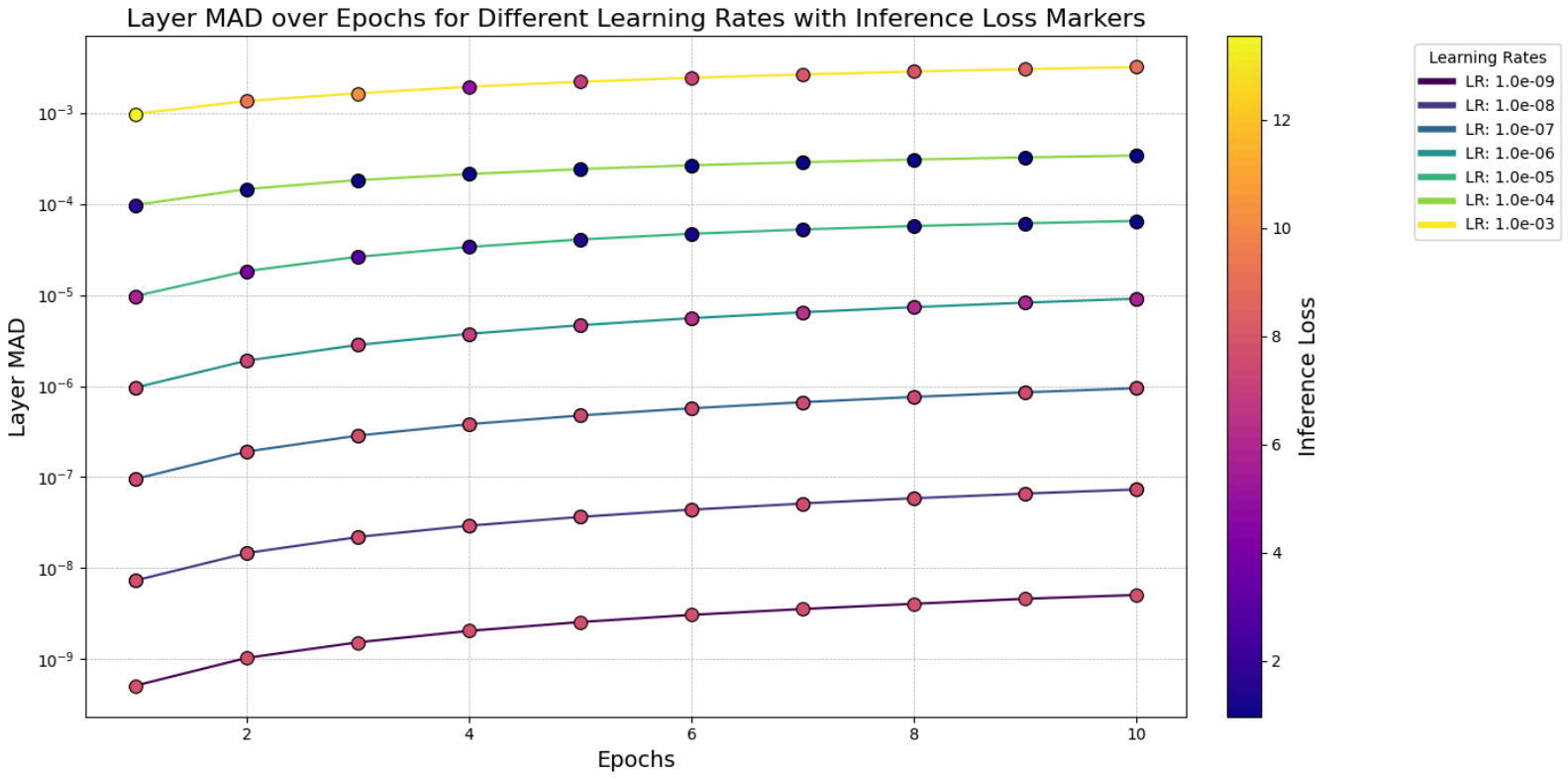
- Mad plots for all learning rates seem the same? Why? The 1e-6 is slightly lower than the others. Still don't have a definitive answer as to why.
- It seems collapse is not linear when learning as seen from results.
- Gradients also are highly effected by learning rate showing a mild decrease. This is probably interesting in that it can be shown that higher learning rates keep gradients at a consistent level implying learning occurs. But lower levels seem to risk vanishing gradients over time? Would require deeper study but it's interesting.

Is it possible that the model collapse is essentially a fixed value if you choose the learning rate on that plateau? Is the plateau always the same for a given data?

7-31-24 (plus some of the above)

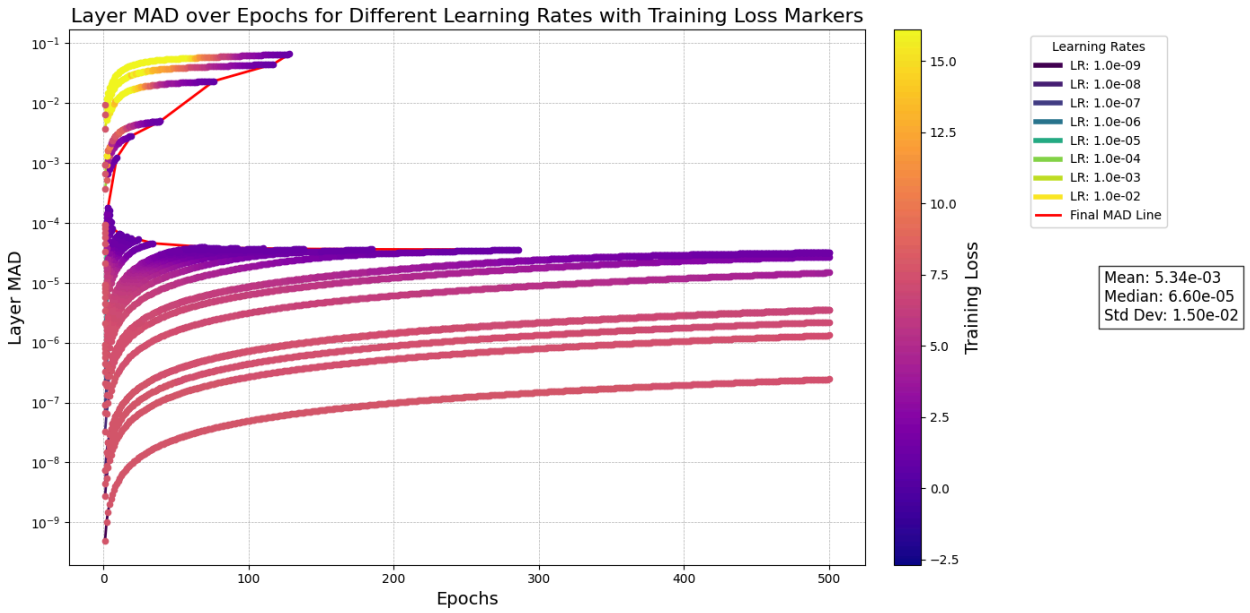
Pareto Front studies:

Current theory is to use a pareto front study to find the absolute optimal learning rate for a model collapse value. With this build a simple LR scheduler that tries to follow the rule.

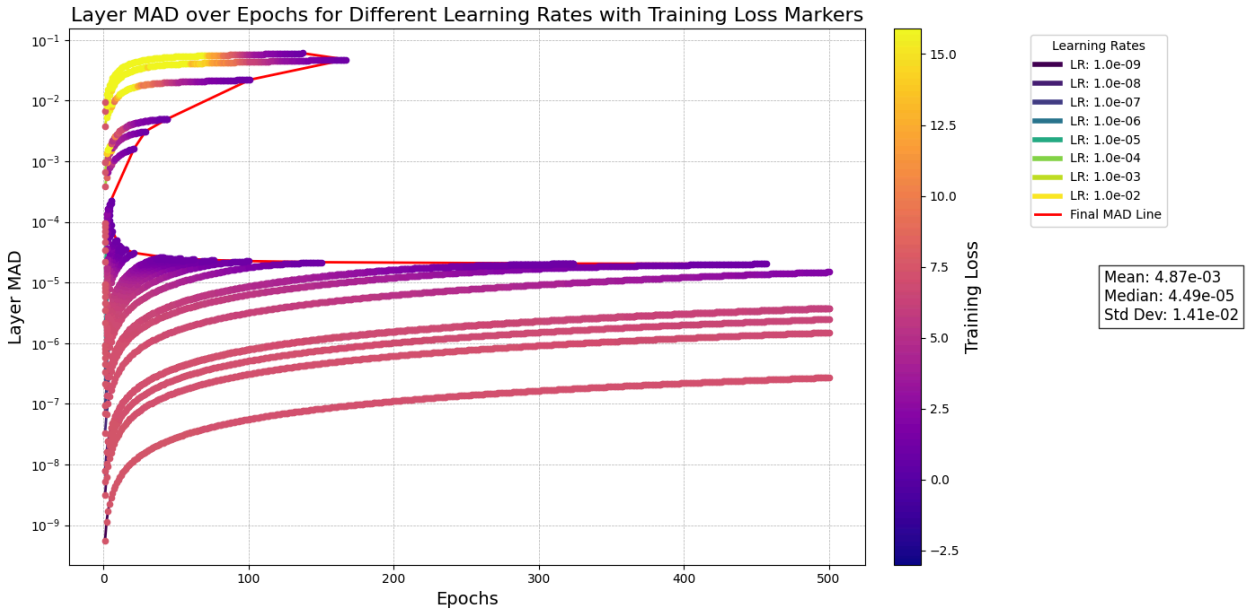


8-1-24: quick pareto studies

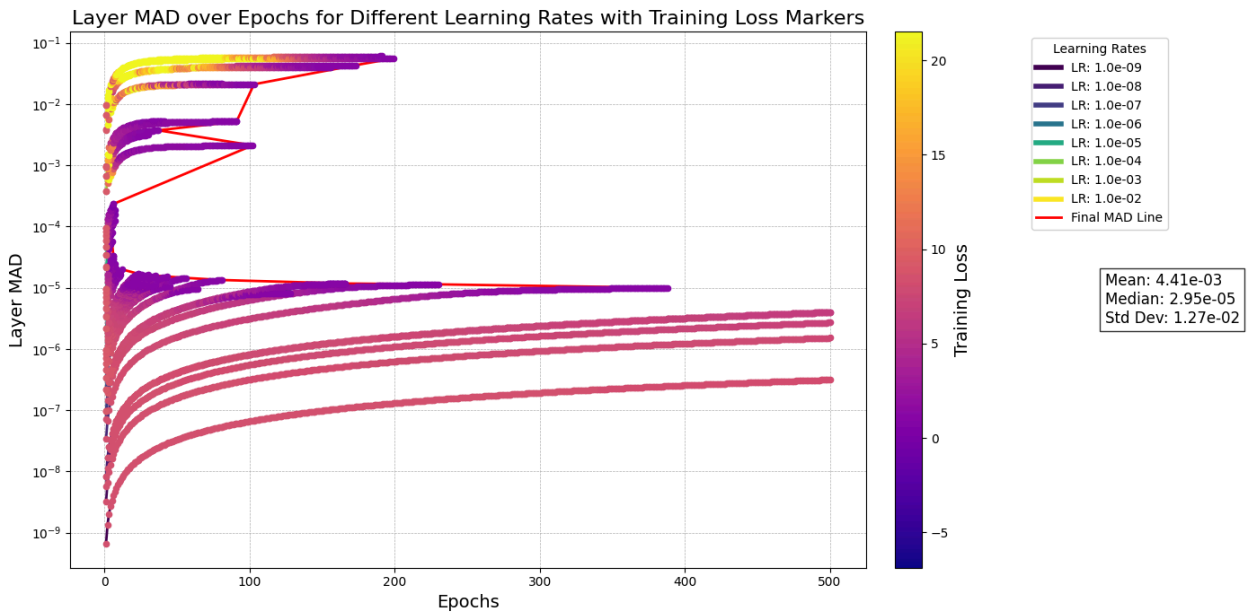
70m



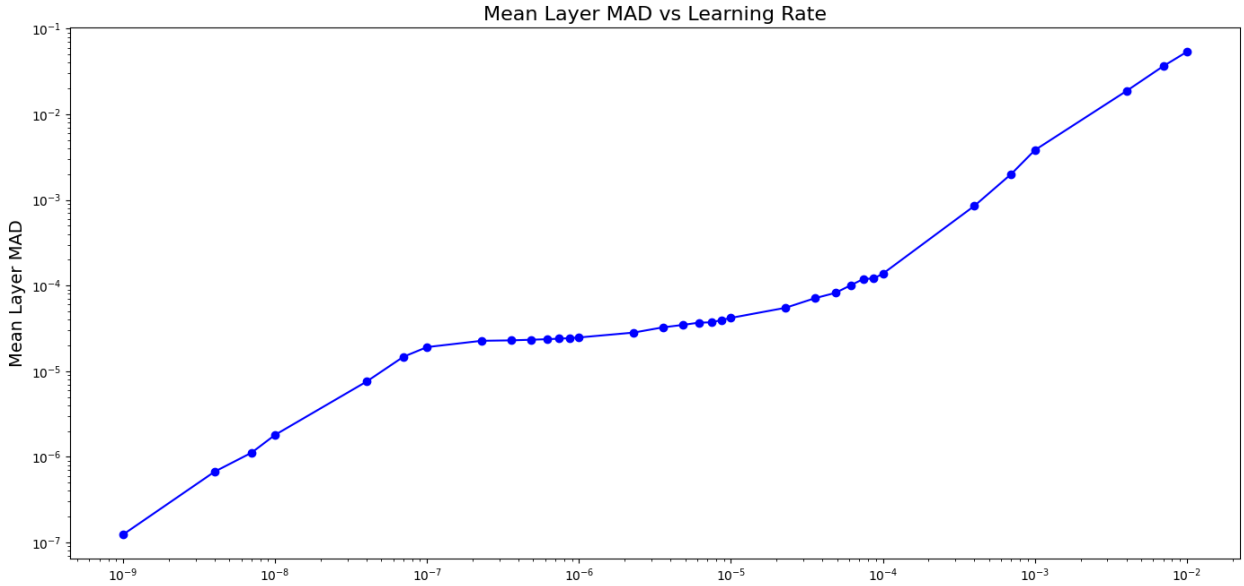
160m



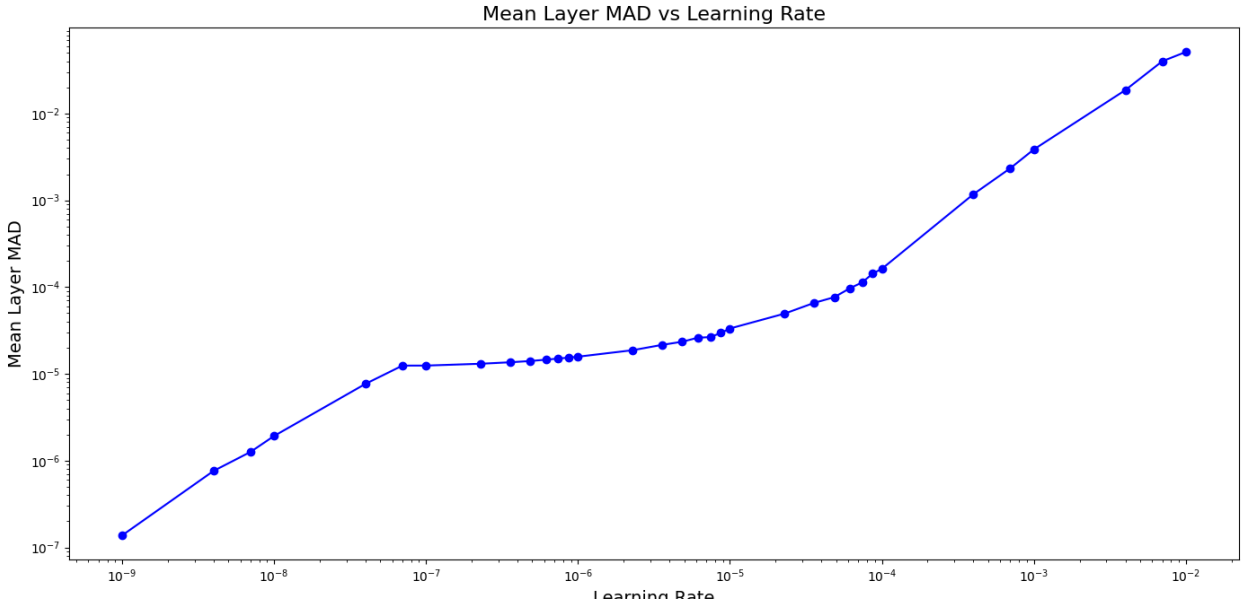
410m



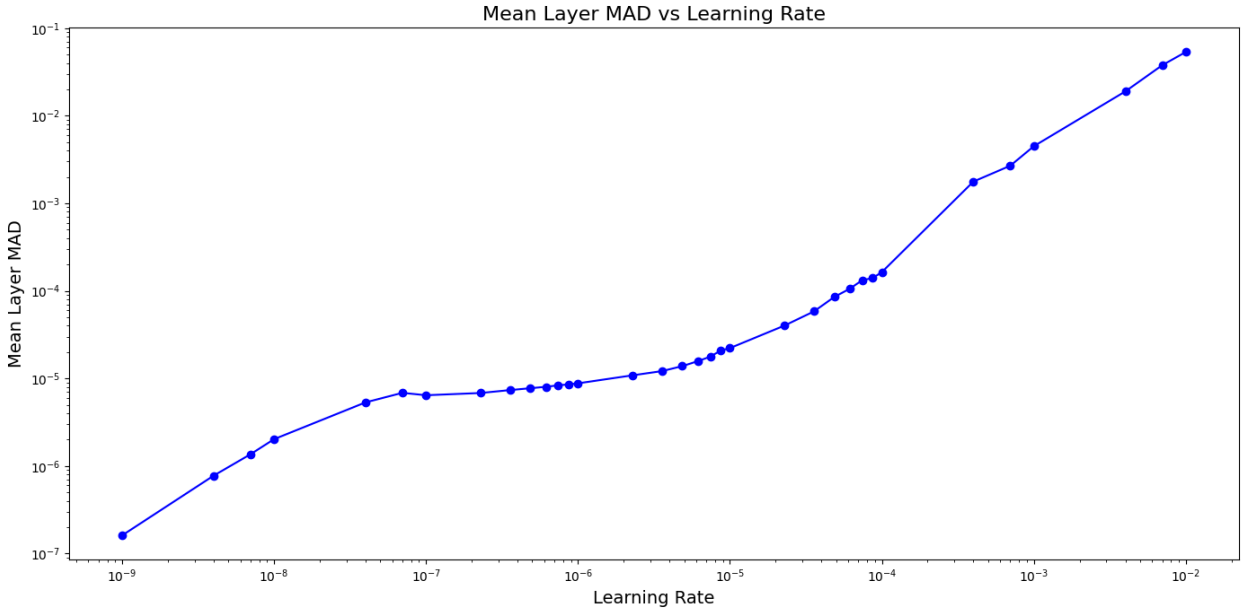
70m



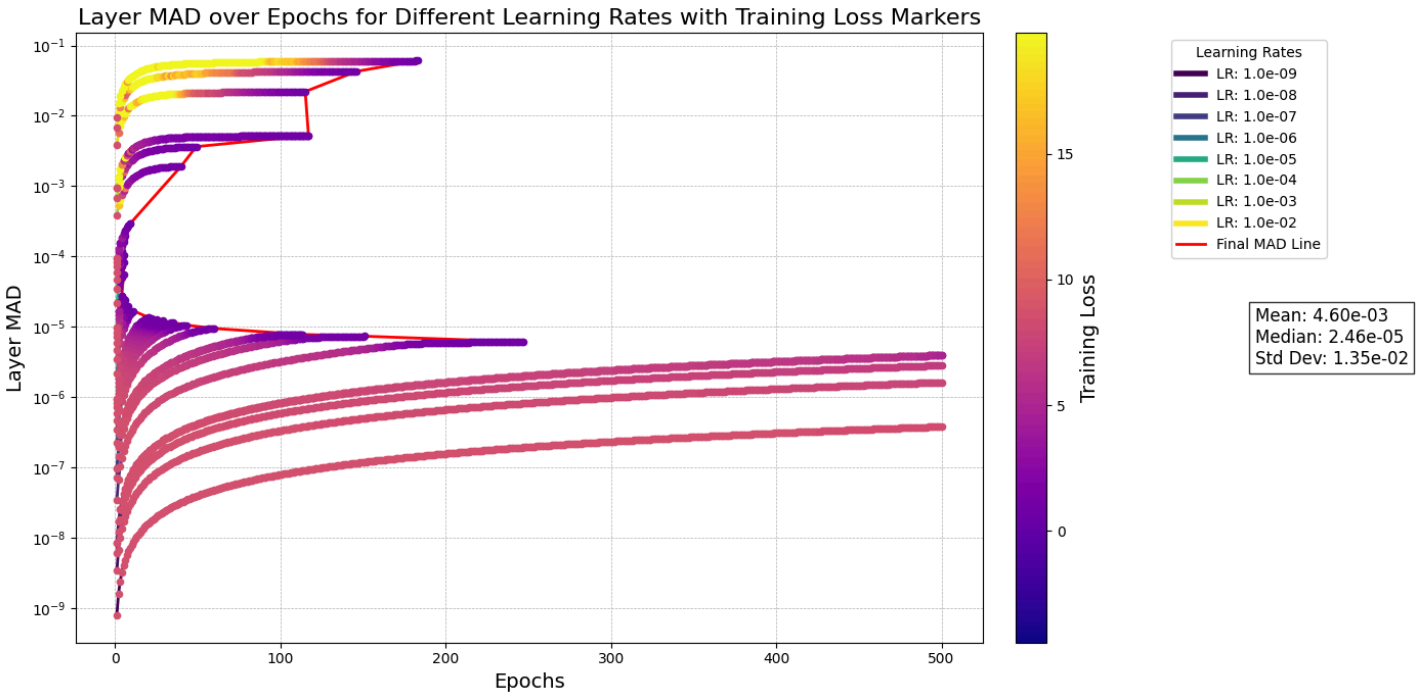
160m

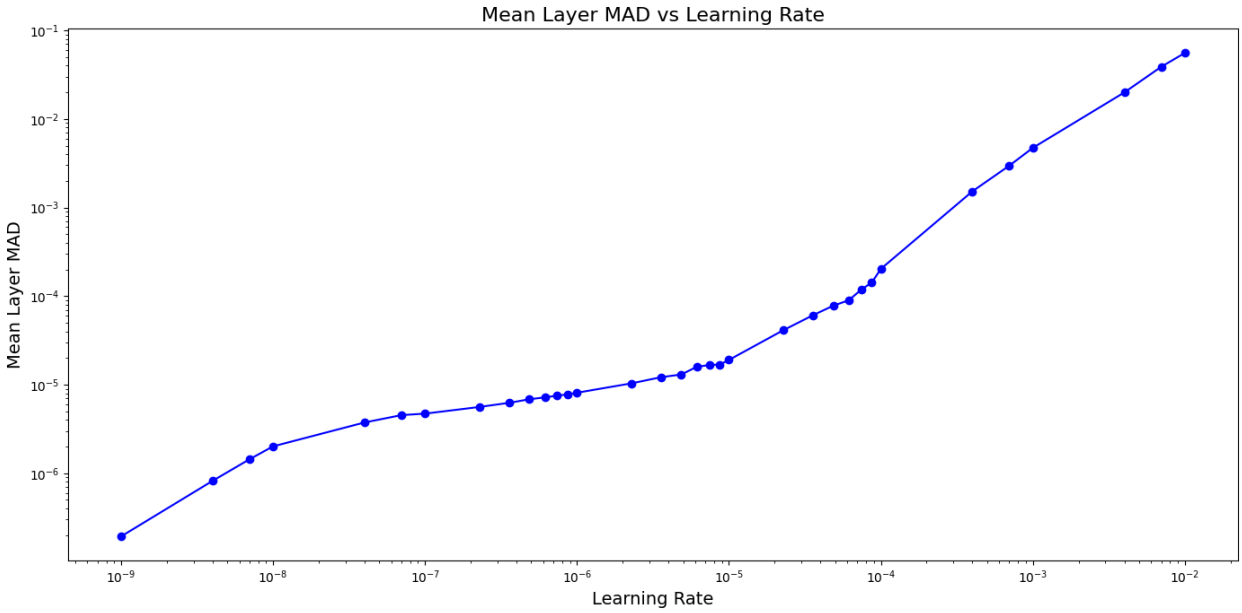


410m



1.4b





Findings:

Larger models are less affected to learn information. The tradeoff is probably that smaller changes to large models have more pronounced damaging effects. The exact correlation of MAD to llm-eval is not 100% mapped out.

Best training method is:

1. Find where on curve you are
2. Train to the limit as fast as possible
3. Slowly train depending on how little model collapse/low loss you want

Ful pareto of DnD (large amount of training)
This needs to be divided into sub sections to better understand system.

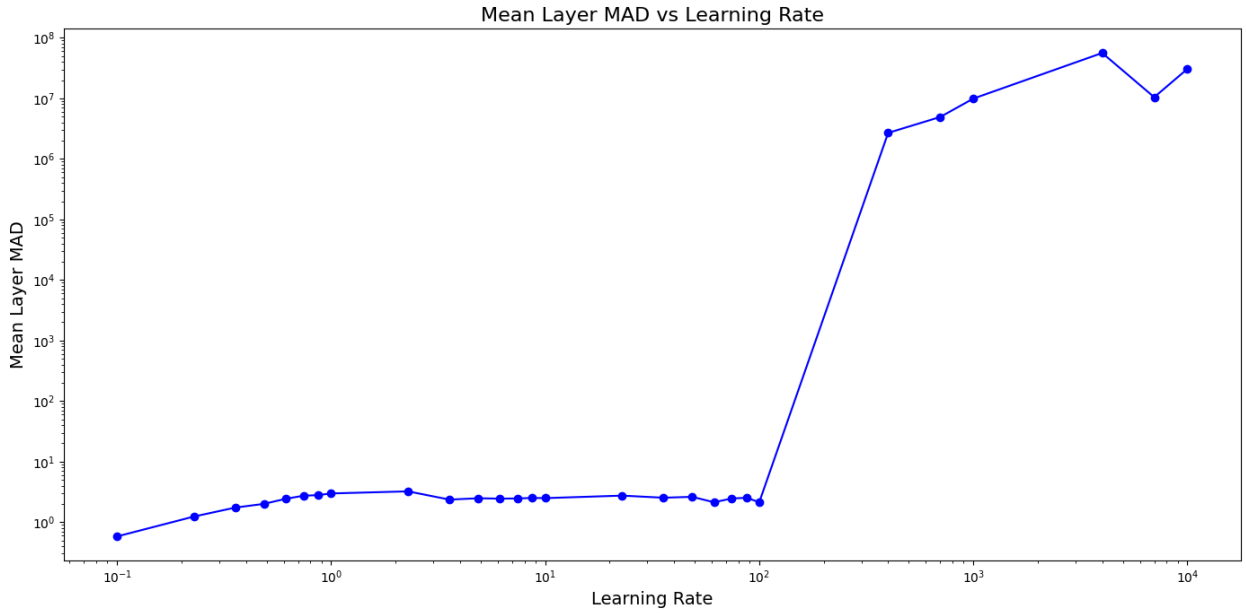
Note, why is there 2 learning rates that never changes mad values??? Looks totally flat?

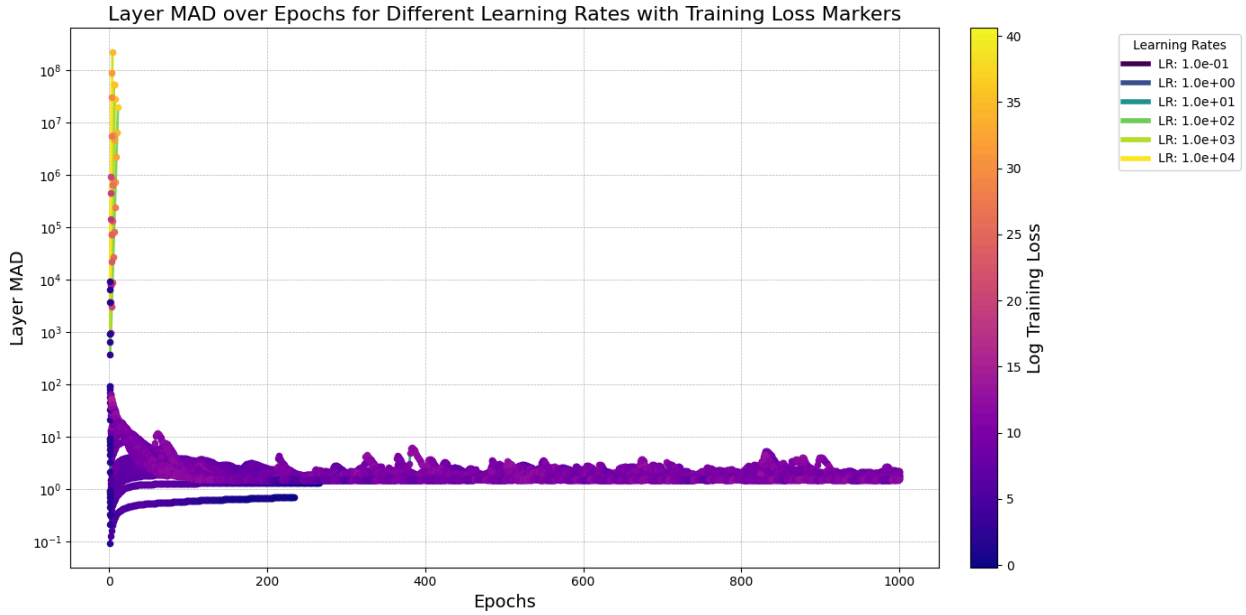
I also find the changing learning rates on the edges of the parabola interesting. Seems that the center is the most stable training space.

Hyper learning rates. Does the top of that pareto curve continue on? It seems sharper in larger models or less slope in smaller. I would like to complete this curve. What happens in this hyper learning rate regime where it still seems to learn just slower and noisier?

Not that exciting basically it works for a bit then explodes. Didn't really see the entire graph but it's as I expected.

It was sort of interesting to see the exploding gradients increase over time.





I'm amazed at something like this. It works, just terribly.

LR	Epoch	Train Loss
3.5714285714285716	759	0.8461510539054871