7-19-24 CL Pile LLMs
7-3-24 LR Studies/Predictors

I am searching for a signal that is sequential memory in llms.

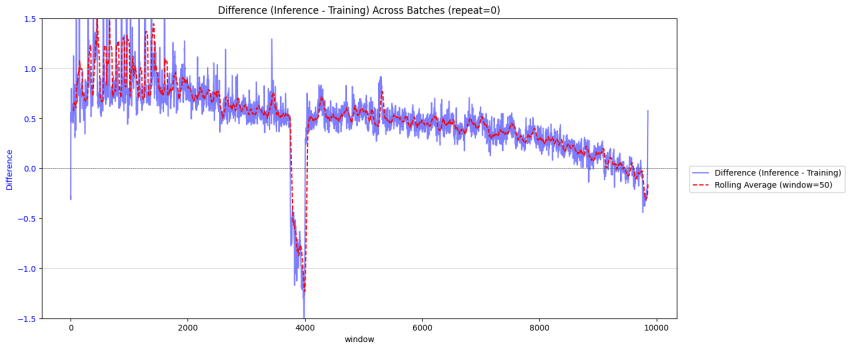
As of now it seems there is no signal at all in any model I have trained. I will most likely need to perform some kind of RL to get sequential memory trained into a model. I will need to do this while also preventing model collapse.

After stopping training the same effect is seen. Sinusoidal memory is still present with strong one or the other memories.

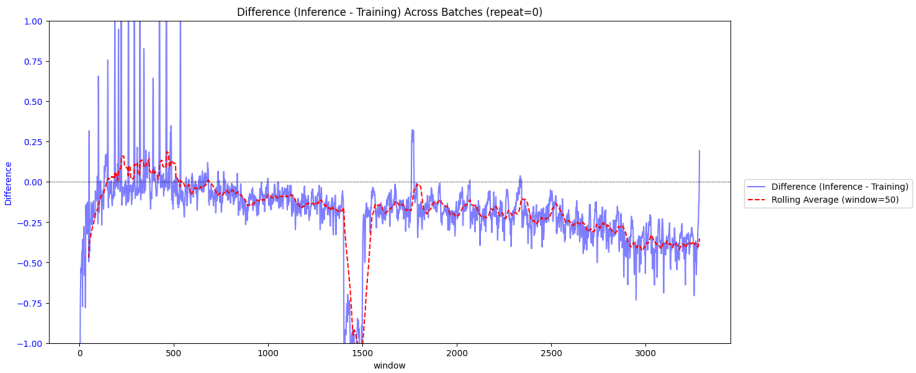
retraining on the rolling window

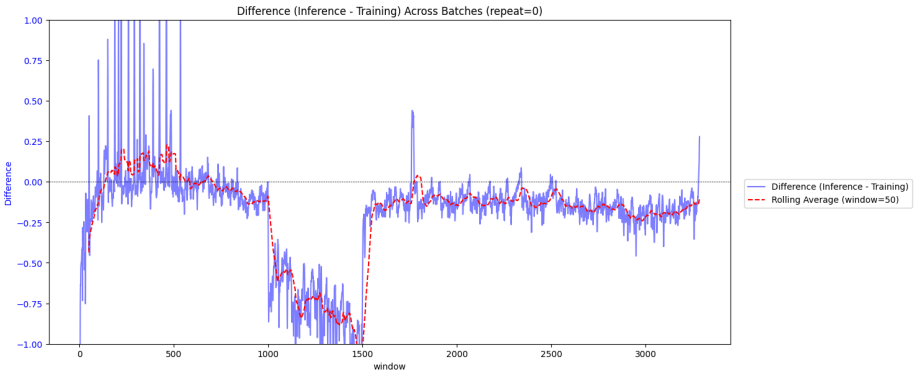
Rolling window 1: retrain on 3000-4000 and 3750-4000
Rolling window 2: retrain on 1000-1500 and 1400 - 1500
Looking for a positive remembrance of data (data in front of the training position reduces loss)

Rolling window 1: no memory signal



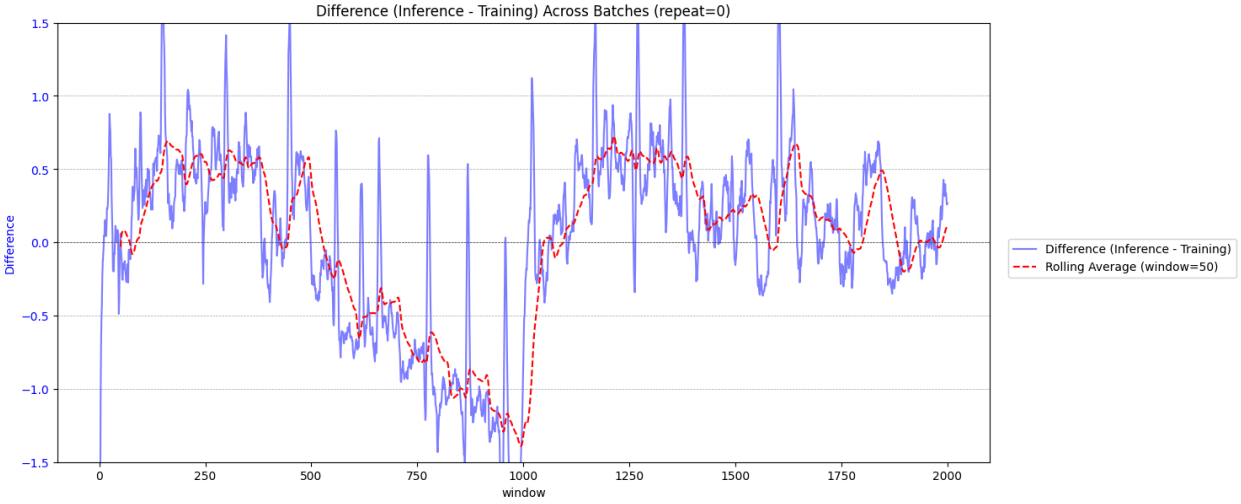
Rolling window 2: no memory signal



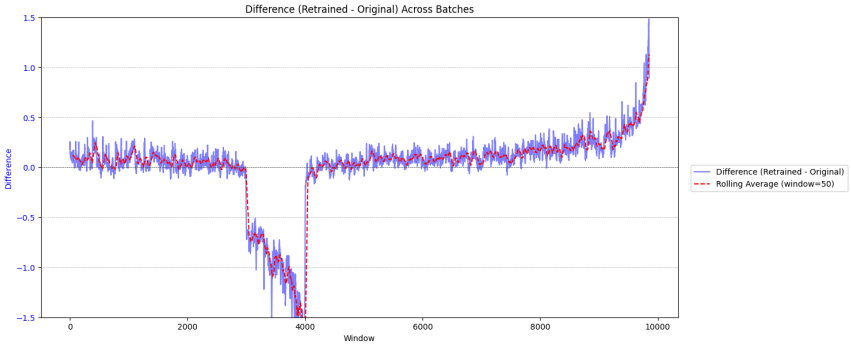


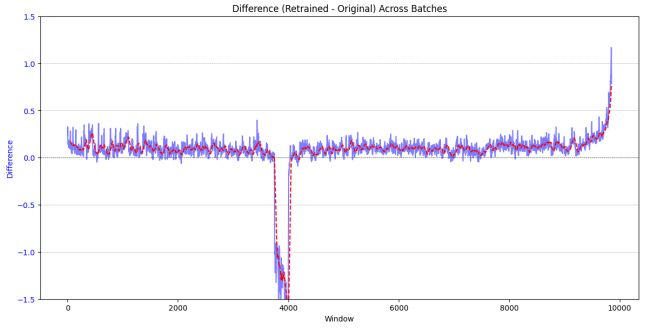
I'm going to train one more model set that puts the difference of inference and training as close to the middle as possible. I am also doing another retraining potion on the 8000-8500 range to see if that shows anything. I will likely use these 3 runs for high resolution testing.

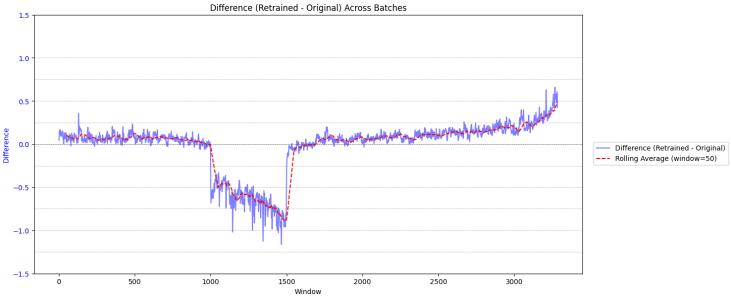
I can't tell. Maybe this worked?

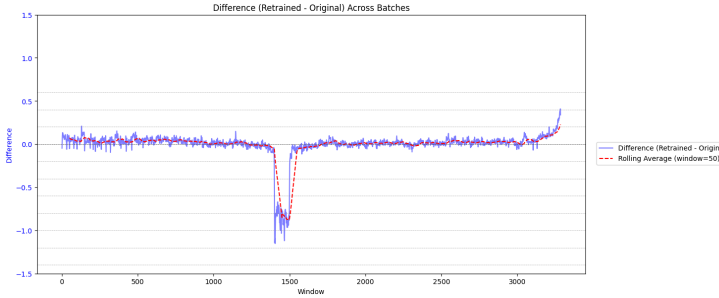


Retrained_inference - original_inference:





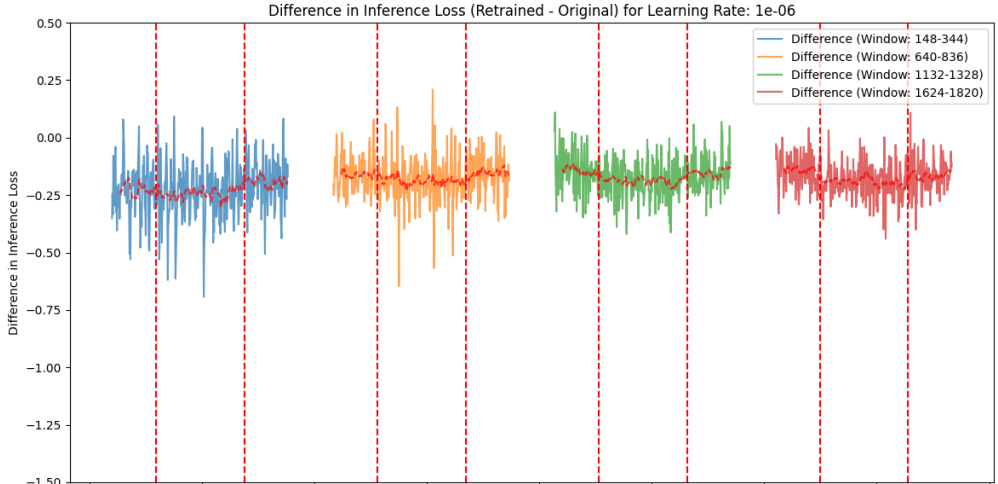




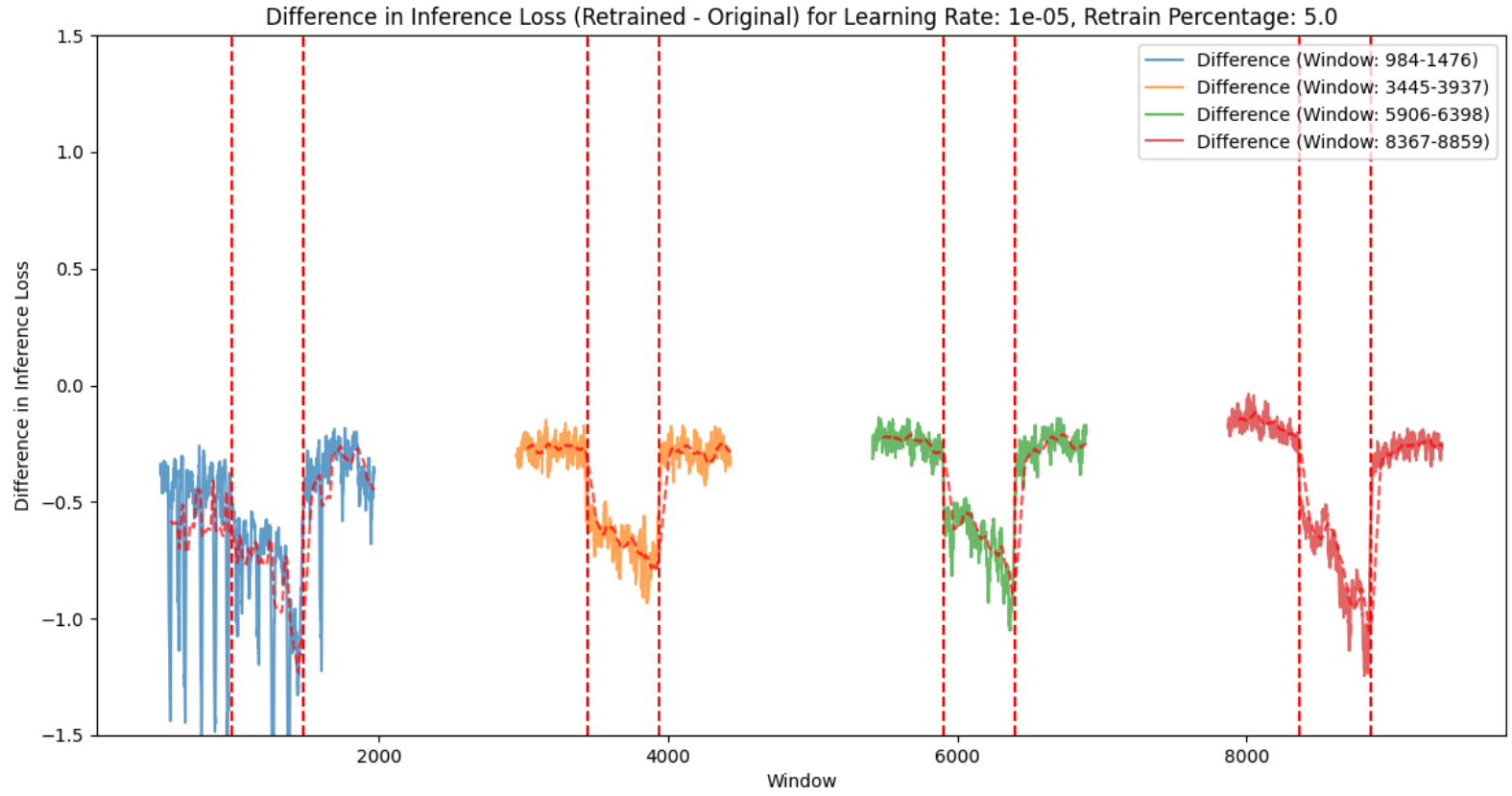
There may be a signal in the smaller window trained model but if it is there it is very small. There is clearly a decrease in the difference right after the training section.

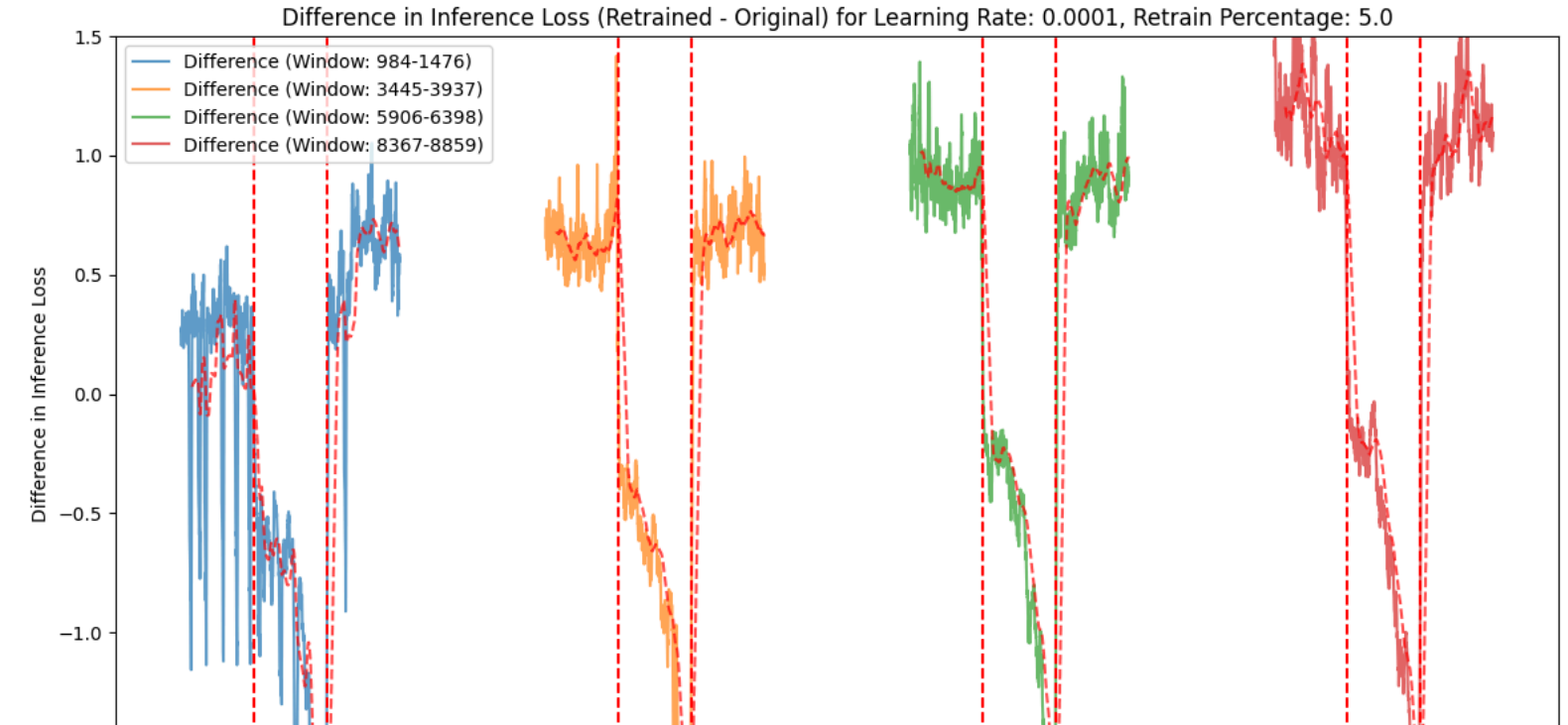
8-7-24
Building a high resolution method and experiment set to try to find the memory signal.

Very preliminary results show no memory signal. But this is a small search and not the original 410m models I wanted to study.



I see no memory effect in any test. This is rw_4 70m with a very long window and small step size. I see zero memory effect at all. Need something entirely different. Maybe there is something there but it is so small I won't know if it is noise or not.





8-19-24: sequential memory pile dataset

Made a dataset that uses previous convo as a seq-seq prediction I want to use this in training. The remaining tokens available will be packed with random pile data.

Goal is to get it to learn with a very large amount of data the previous discussed text.

The sizes vary per example.

8-21-24: Lots of debugging and seq2seq combined with AR

Preliminary results show that a standard training method with a base standard learning rate is a small change between previous results and new in training.

Created a new LR gradient method. This method shows large promise in sequential memory.

Standard LR has ~0.05 -> 0.1 decrease. This method show ~0.1 -> 0.3

I'm interested how this LR profile can effect this.

I'm stopping this experimentation to focus on actual chatbot memory formation.

I don't see how my standard models will succeed at all. I want to use actual chatbot tuned models.

-> I couldn't convince myself that a chatbot could do it and a simple model could not

8-27-24: sequential sentences training

Training a model on unique sentences and then testing if it can predict the last word of that sentence. I find that it can in a rolling window method however it isn't "strong" enough.

This makes me think that the main problem I will face is that forming a memory is so inefficient that it prevents and sequential knowledge to be formed.

For example. This below is predicting the last token and the last word in a set of sequential data. The key thing to note is that the token prediction is just the last token of a sentence. It can remember and predict it quite well over time. 85% in a rolling window is pretty strong. However the word prediction is poor.

This seems to indicate that the farther you try to stretch information from source the harder it is to remember. This is also what was learned in the previous study that training on examples does lead to related questions being learned as well however they were markedly lower accuracy. There seems to be a "distance" function in model memory that needs to be improved.

Token Prediction: 85/100, Word Prediction: 17/100

So how could I improve this above? This is basically what I need to improve in order to achieve the sequential memory. My main idea is around creating a pseudo statespace in these models.

1. Have in context information for the model to see and try to learn from (the rolling window)
- This will be like the context kept in a agent chatbot
- I assume it will be better at answering these questions than before

2. Use variable learning rates to create multiple different statespaces within the model.
    - Let's go off the idea of before where we have a certain percentage of parameters train on differing learning rates.
    - The idea being the higher learning rate parameters can act as a inbetween of in context information and the memory.

I could find a "coupling" factor by slowly decreasing the amount of context included in a prompt and trying to keep a Q/A accuracy the exact same.
So this is to say that Given the context and answer quality from the beginning (given the sentence can repeat it accurately) Can I keep that score the exact same as it moves into long term memory?
The idea being that I'll use in context information as the first layer of state space and then the high learning rate parameters and then the onto the lower learning rate parameters. Effectively coupling in a smooth way data from in context to in parameters.

Would I need a new attention method? Focusing on the most recent text over the previous?
Using the entire rolling window put into context and different lr percentages:
Epoch 1, Step 52, Train Loss: 3.153475284576416, Token Prediction: 98/100, Word Prediction: 24/100

Much better performance than before. But I don't think the context is helping at all.

Token prediction is heavily influenced by the amount of pile data vs window data. However I don't see that direct correlation with word prediction? Had a run where the first inference came back with 11/100

This is also pretty random due to the lrs and choosing random parameters to have what learning rates

9-2-24:
Studying next word prediction with helping context and rolling window training.

Seems a bit sporadic because of the random learning rate selections.

Also notice that increasing the repeat sentences can seem to hurt the responses if there are too many, unknown why.

Nearing the end of training the models with context completely fell apart. Significant decrease in predictions occurred even worse than the no context model.

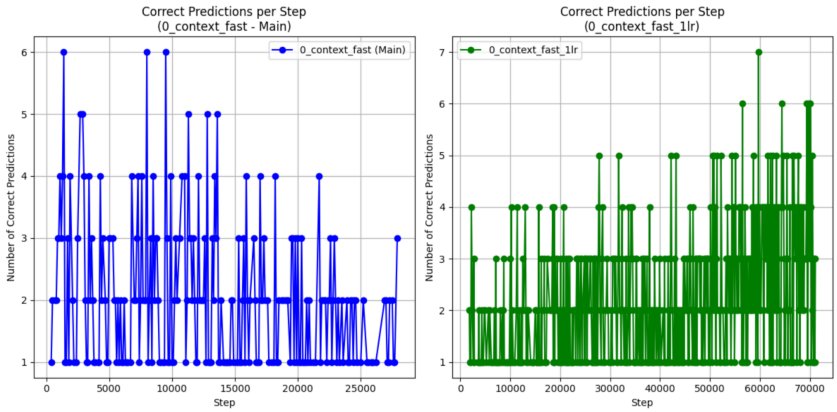
While the 0 context model was false it's responses were still somewhat probable. The higher context models gave a lot more propositions like your my I his.

Could this be because I didn't put a good dividing token between sentences?

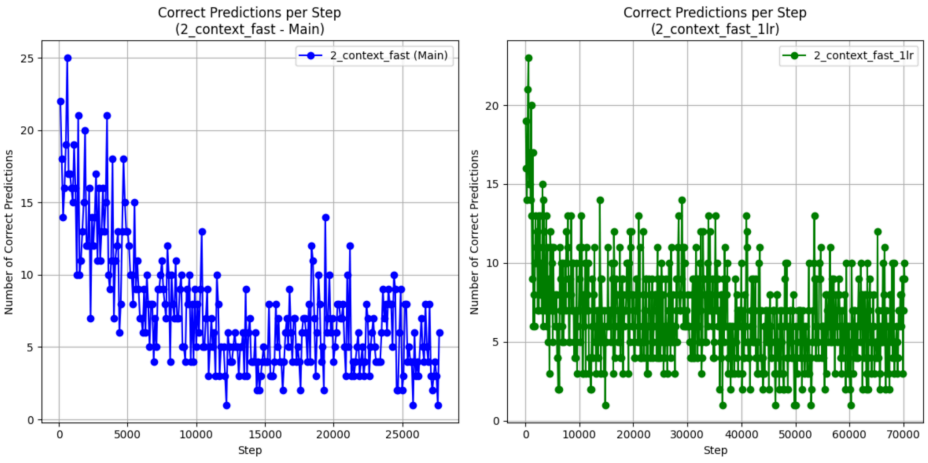
9-6-24:

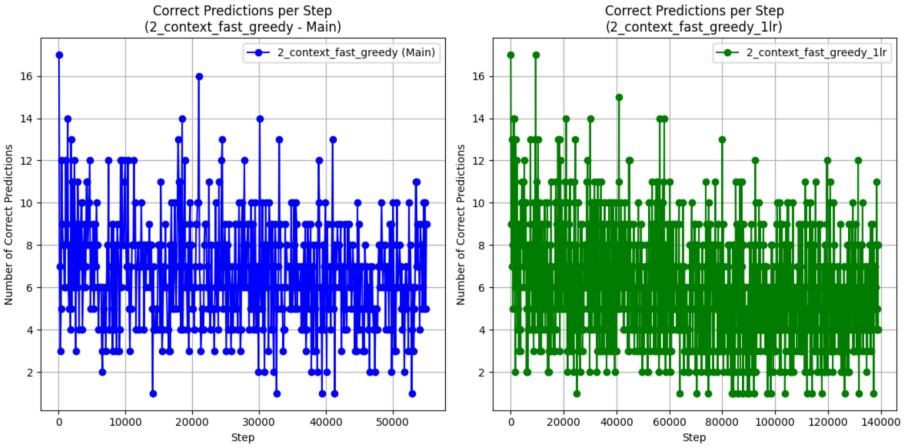
Finding from yesterday showed that multiple learning rates do no significant advantage to the run at all. They do however massively slow down the training steps by >50%.

- Results also seem to show that with zero context the 1 lr performs better in the long run and multiple learning rates perform better in the short run.



Top_k sampling vs greedy and less window methods





Greedy with a smaller max tokens sems to damage earlier performance? I don't know why it decreases so much but that is clearly a problem. It should be atleast stable.

9-6-24: thoughts Overall

So it seems from the research out there and my own thoughts that one of the largest issues is that even if you perfectly achieve continuous learning there is a saturation of parameter memory that occurs. Adding layers is an infinite process that may work but it still means you need infinite scaling.

I think my chaining method may be more useful than I originally thought. Being to grow the dimensionality of a model may be the key. Let's say even if I solve this episodic memory problem. Without a continuous growing something the system will saturate.

A growing a new connections followed by a pruning may be required. Grow the number of new chaining connections allowing completely unique architectures to evolve followed by a pruning method.

I'm going to look into saturation and improving accuracy of a model at a constrained parameter size.

With this I will then take this evoliutionary growing model and apply ewc to it. Training on new information while protecting old and adding new connections.

Might be able to do that in a month?

9-8-24:
Looking back at this work one of the keys I am completely ignoring is that I don't actually know the exact requirements for an LLM to learn from a single example.

Even more importantly I don't know what's needed for an LLM to learn from a single example when there is a batch of information like what I am doing above.

Another point is that I don't know what's needed exactly to transfer that information to the next step correctly.

-> I need to go back to the original LR methods and build out something to actually test this in great detail.

NOTE: OPT models are a suite of scaling LLM models that fo from 125M to 175B

PAPER: Episodic memory and memory retention:

1. **Episodic Memory Retention**: How well can LLMs learn and retain new information after being exposed to it once or with minimal repetitions?
2. **Scale and Model Size Impact**: How does the size of the model affect its ability to remember new information and not forget prior knowledge?
3. **Forgetting Dynamics**: How does the model's memory decay or persist over time during continuous learning? Can you mitigate this forgetting with specific architectures or training strategies?
4. **Integration of Old and New Knowledge**: Explore how to prevent catastrophic forgetting while ensuring that new information is incorporated seamlessly.

Experiment 1: Baseline Episodic Memory Retention

Understand how well LLMs of different scales can **retain knowledge** from a **single training exposure**.

1. **Data**: Use **Data 4 (hard-to-guess questions)** to introduce **single examples** of novel information. This dataset is perfect for testing retention since the training and test questions are unrelated but thematically similar.
2. **Procedure**:
    - Train LLMs (from **Pythia suite**: 1B, 2.7B, 6.7B, and possibly 12B) on **10 training questions** from Data 4 for **one epoch**.
    - Immediately test on the **10 held-out questions** to measure **initial retention**.
    - After training, continue fine-tuning on an unrelated dataset (like **The Pile** or **CR3**) for 10 epochs and test **episodic memory** retention over time by periodically re-testing the 10 held-out questions.
3. **Metrics**:
    - **Initial accuracy** after training on the 10 held-out questions.
    - **Memory decay rate**: Measure how much accuracy drops after each epoch of fine-tuning on unrelated data.
    - **Model size correlation**: Does episodic retention improve with model size?

Experiment 2: Episodic Memory in Mixed Data Batches

Assess how well LLMs can retain episodic memory when new information is introduced alongside **irrelevant data** during continual learning.

1. **Data**: Mix **Data 3 (synthetic sentences)** with a base dataset like **The Pile** in a **50/50 batch split**.
2. **Procedure**:
    - Introduce **10 new synthetic facts** (from Data 3) in a single batch while training the model with **The Pile**. Test whether these facts are retained over time by periodically checking accuracy.
    - Test on these synthetic facts at multiple intervals (e.g., after 1, 5, 10 epochs of fine-tuning on the rest of The Pile).
3. **Model Sizes**: Use the same **Pythia models** (1B, 2.7B, 6.7B) to compare how different model sizes handle **interference** from unrelated data.
4. **Metrics**:
    - **Retention over time**: How quickly does the model forget or retain these 10 synthetic facts?
    - **Impact of Model Scale**: Larger models are expected to handle the interference better.

Experiment 3: Long-Term Memory Retention with Continuous Learning

Evaluate how well LLMs retain specific facts over long periods of continual learning, simulating a **real-world continual learning scenario**.

1. **Data**:
    - Use **CR3** as the continual learning dataset, which reflects a more realistic, long-running task where context shifts.
    - Inject **single, specific facts** (like from Data 4 or Data 3) intermittently and measure how well these facts are retained during long-term learning on CR3.
2. **Procedure**:
    - Begin training with **CR3** data, introducing **specific facts** after every few epochs.
    - Measure how much **new facts** affect the retention of **older facts**.

    - Test retention periodically by querying facts introduced earlier to see if the model is still able to recall them despite continual training.
3. **Model Variants**: Test on Pythia models with varying scales (1B, 2.7B, 6.7B).
4. **Metrics**:
    - **Long-term memory decay**: Measure the model's accuracy on early-injected facts after each training epoch.
    - **Impact of continual learning**: Does mixing new information disrupt the model's ability to retain older facts?
    - **Scale correlation**: Larger models should exhibit **slower forgetting rates** compared to smaller models.

Suggested Outline for Your Paper

1. **Introduction**:
    - State the importance of **episodic memory** in LLMs and **continual learning**.
    - Highlight the novelty of studying **scaling requirements** for memory retention.
2. **Related Work**:
    - Reference scaling laws, continual learning, and memory-augmented models.
3. **Experiment 1: Episodic Memory Retention**.
    - Describe how single-instance learning is tested.
    - Report results and analysis of model scale and memory decay.
4. **Experiment 2: Episodic Memory in Mixed Data**.
    - Present how episodic memory performs in mixed-data batches.
    - Show interference rates and model size comparisons.
5. **Experiment 3: Long-Term Memory in Continual Learning**.
    - Report long-term retention of facts across continual learning on CR3.
    - Analyze memory decay and the impact of model size.
6. **Discussion**:
    - General insights on how scaling impacts memory retention.
    - Practical implications for building **LLM agents** in long-running tasks.
7. **Conclusion**:
    - Summarize key findings.
    - Discuss future work, such as incorporating **memory-augmented architectures**.

9-13-24: Results experiment 1:
So I ran experiment 1 here and found some unusual activity. It seems that LLMs of multiple sizes and types have a latent memory mechanism.

It seems that when training on an initial item and then continuing to train the model actually continues to train on previous information. The scores improve as you train on unrelated different information.

The information seems latent and can sometimes hide from scores for many steps of training.

I was thinking this mechanism might be the key to what I'm looking for. What if data inside an LLM never really leaves it simply can't be accessed at the time?

I want to run an experiment similar to this one where I continually loop a selection of data. This selection of data would then have an insert of information into it at a certain point. I will then monitor its memory score throughout the loop of training.

Is it possible that looping information is a strong way memory is stored inside a model? As in, is it possible that an LLM actually does not remember anything until the right circumstances arise?

If I build a looping data that the LLM continually trains on. Inserting information at a certain time. Would the LLM forget it everywhere but the location it needs to remember it?

Basically I am trying to find a way to use the results of my test in a real world scenario. If it is true that LLMs lock away information that they can bring back out again at later periods this may be the key to structured memory.

The idea would be to create repeating or slightly repeating loops of information. Within these loops I insert new data. The idea being that the repeated nature of the loop allows information to be stored and retrieved in the locations at the proper times and when needed.

If a model can slightly remember the loop would it allow the model to store more information? As in in a repeating loop it would attach memories to certain parts or certain loops. While it wouldn't remember it outside of the loop it would be able to recall it then.

Exploiting the latent knowledge effect seen in experiment 1 may be the key to long term sequential memory.

It also may imply that memories are literally non-existent until the right set of events arise. While the model would score poorly on memory tests as a whole it's RELATIVE memory scores would be good.

This may be the way to do sequential memory. Instead of remembering everything that's ever happened all the time it only remembers the relative information.

9-14-24:

In regards to using optimizers as context and loops.

What if I had a special loop just made for learning new information?
https://arxiv.org/abs/1703.03400

Say I have multiple loops I train on.

Let's take a few and average out their adam optimizer moments. This acts similar to this model in the paper trying to learn many things more effectively.

In the paper they get the gradients of multiple tasks for one model then average them for an update. This allowed quicker adaptation.

So what if I did this but for loops.

I would make a "new information loop" This adam moments would be the averaged of all of my loops. Run the loop and at the end take that average as well for the new information loop optimizer. And then move this new loop to use the optimizer moments at then end of that loop for a new loop creation.

1. Have loops and multiple optimizers and their moments as "context"
2. When a new loop is to be added
3. Average all current loops moments
4. Train the loop on the new data
5. The final optimizer moments are now the new loops "context"
6. The new information loop now has another set of optimizer moments to average the next time a loop is to be added.

Experiment Design 1 Memory Loops:
1.I need some help designing a new experiment for a paper. So make sure in your experiment design you incorporate details like what exact data and plots I will need.

2. I want to build "memory loops"

- A memory loop is based off previous work that found LLMs have sinusoidal memory.

- I am trying to exploit and expand on this idea.

- The current idea is to see how these sinusoidal memories can be controlled

3. I want to build sections of data with a specific memory data inside of it. This data is general and the memory data is not guessable and quite random. This is if the LLM responds I know it learned that information

4. What I want to do is train on a loop of information.

9-16-24

“loop_training_experiment_1”

So I made a memory loop method but found unexpected behavior

Loop 0:
- loop 0’s qa data
- loop 1’s qa data but with loop 0’s answer

Loop 1:
Same but opposite

What’s unusual is that while I am seeing the effect of either qa 0/1 being correct or incorrect what’s unusual is it seems to have the opposite effect I intended.

It seems that when starting loop 1 qa 0 is highly correct, when starting loop 0 qa 1 is highly correct which is the exact opposite effect I expected.

This behavior changes over time though so perhaps this is an intermediate effect?

I think I am seeing the effect I wanted. However the model might have no ability to remember or associate time. So it randomly decides to either swap or flip the qa 1 or qa 2 parameters around internally.

However it might not be associating the pile data with the parameter swap needed as it doesn’t know the order it occurs in at all.

I want to repeat but I want to watch the optimizers. And perhaps I will push the optimizers around for testing. First I will test what happens when training stops

Loop_training_experiment_2

- Found similar results.
- Even after turning off training the model remembers for a very long time in the same fashion. Clearly some decay but the information is very stuck in the model and will come and go over time as expected

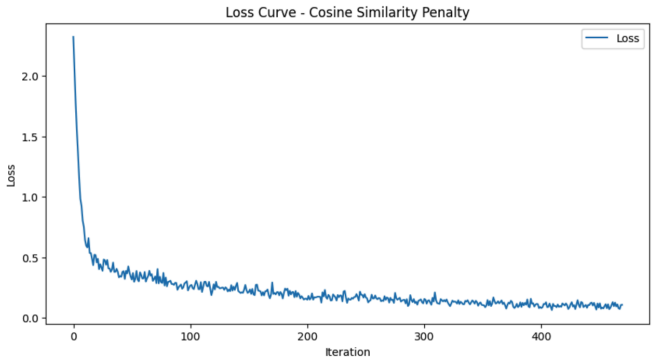
NOTE: This effect is stronger when using ADAM optimizers but it is still seen in SGD methods just not as often. The initial decrease is always seen though. The later revival of the information is seen less often in SGD which is expected but it does happen.

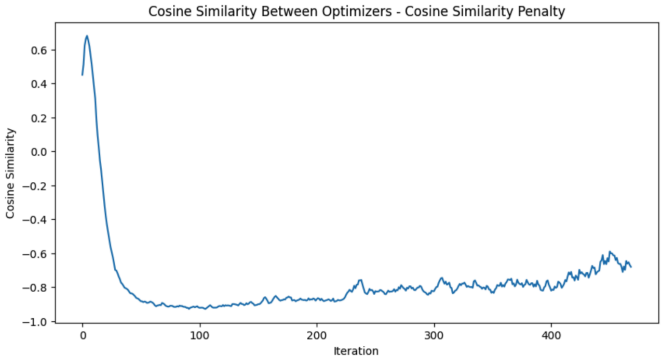
9-17 Optimizer work

Looking at different optimizer repulsion methods. [Based on this chat]

- Images below are on a test set to just look at what these optimizer changes do

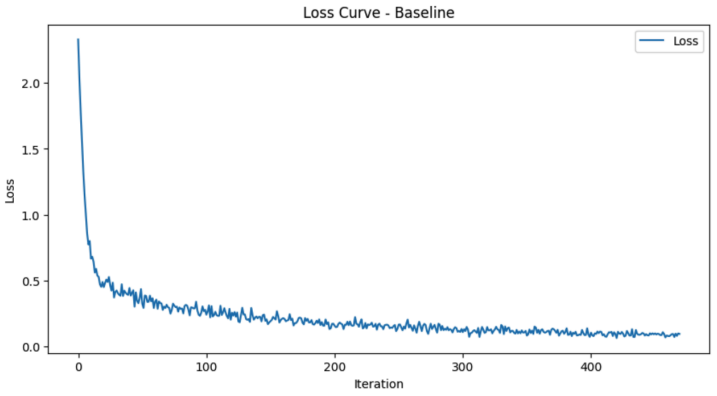
Analysis for Cosine Similarity Penalty:

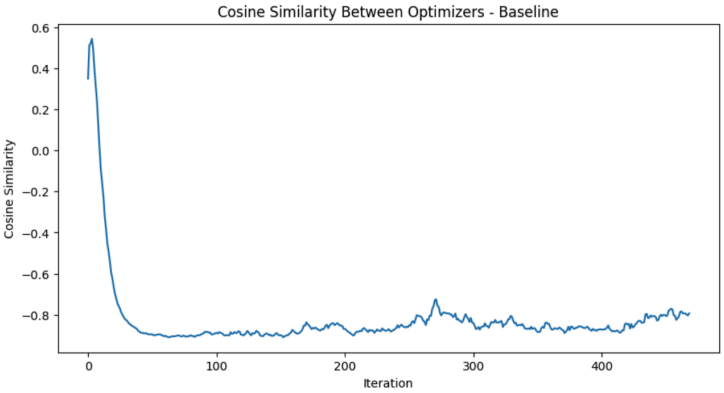




Average Loss: 0.2230
Average Cosine Similarity: -0.7706
Test Accuracy: 96.73%

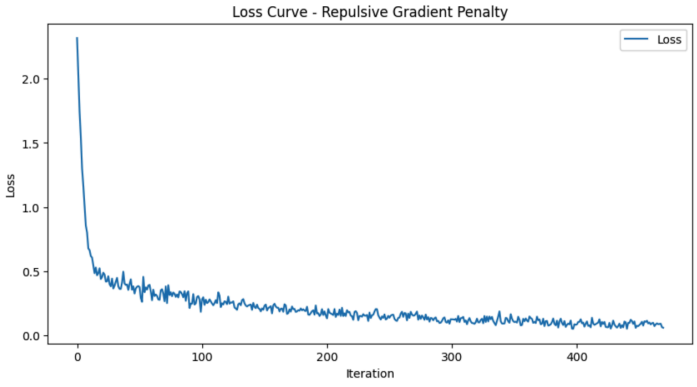
Analysis for Baseline:

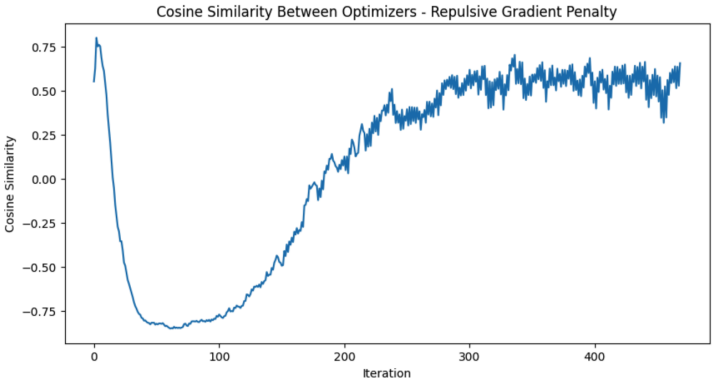




Average Loss: 0.2162
Average Cosine Similarity: -0.8230
Test Accuracy: 96.90%

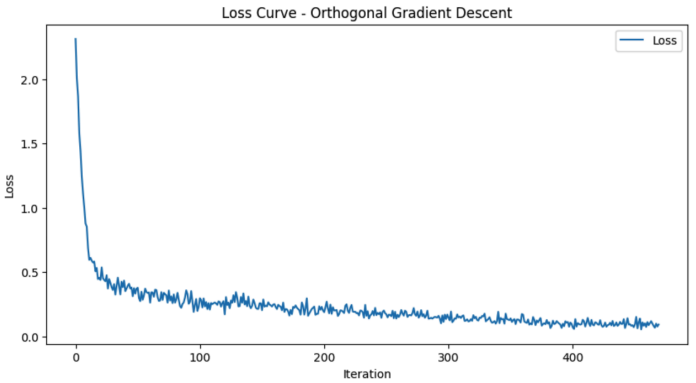
Analysis for Repulsive Gradient Penalty:

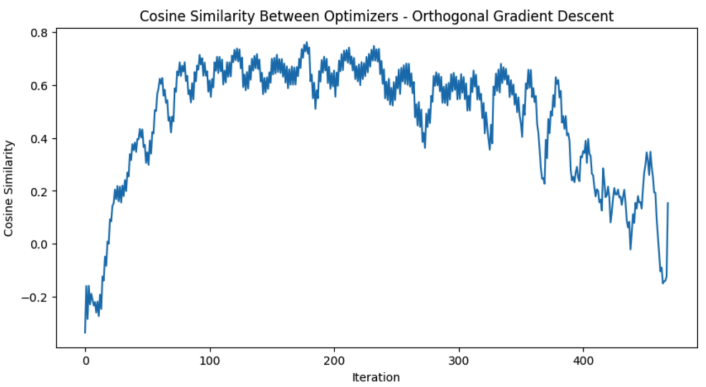




Average Loss: 0.2157
Average Cosine Similarity: 0.0792
Test Accuracy: 97.13%

Analysis for Orthogonal Gradient Descent:





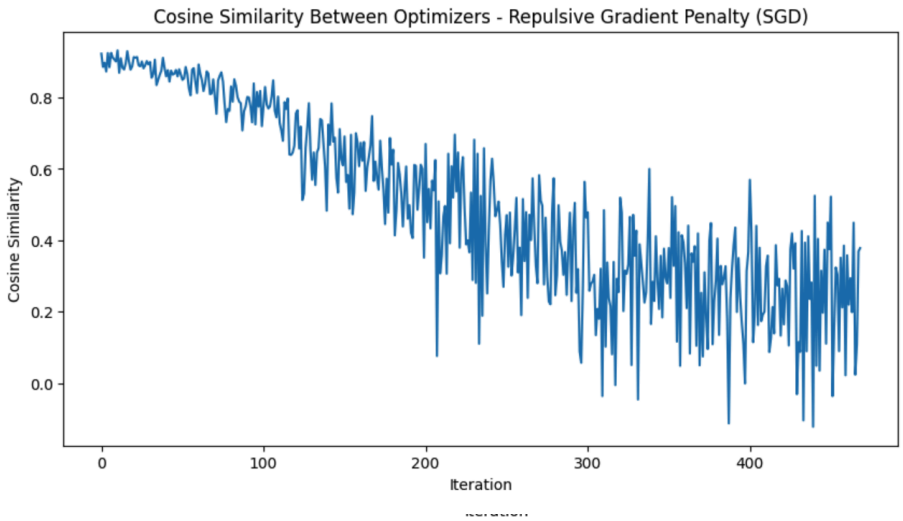
Average Loss: 0.2295
Average Cosine Similarity: 0.4717
Test Accuracy: 96.81%

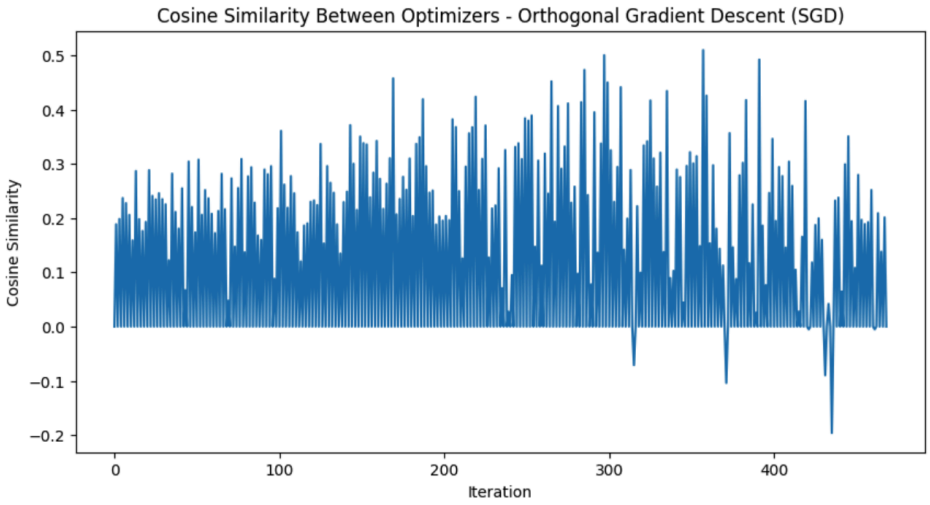
1 means the vectors are exactly the same (pointing in the same direction).
0 means the vectors are orthogonal (completely different in terms of direction but not opposite).
-1 means the vectors are exactly opposite (pointing in completely opposite directions).

I find it interesting that the orthogonal method seems to be converging to be opposite direction but not the opposite direction?

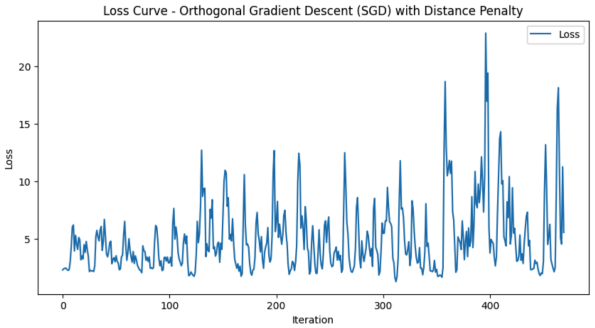
Chat says: This can happen because orthogonalizing gradients directly in deep networks is challenging. The moment-based nature of Adam can also make gradient orthogonality less effective over time due to its accumulation of historical gradients.

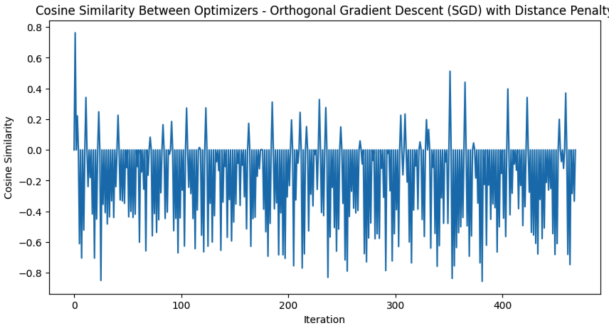
This was a repeat withSGD instead of adam to remove the effects of the moments.



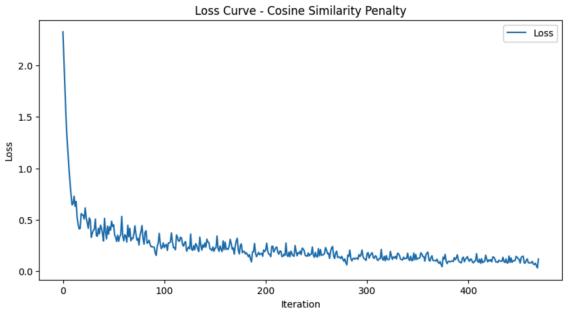


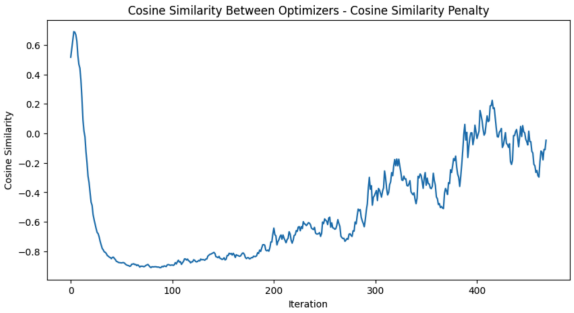
Shuffle was on, repeats below
-almost identical results, the ones that diverged are below



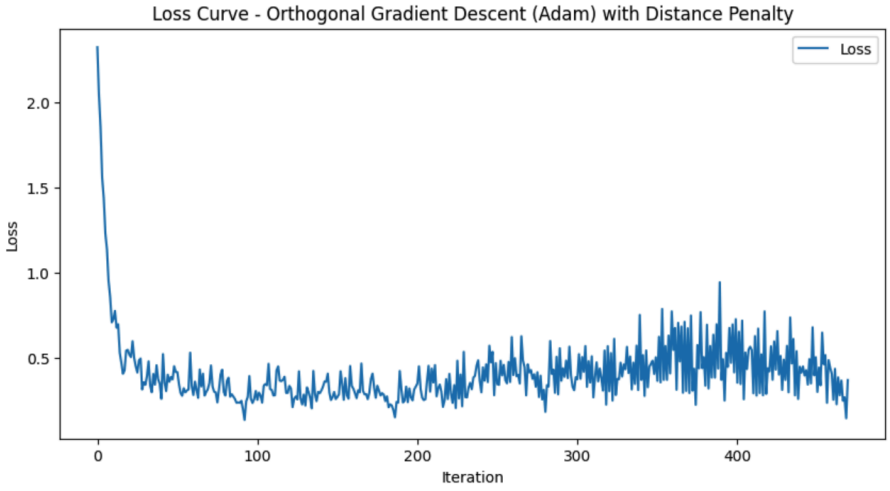


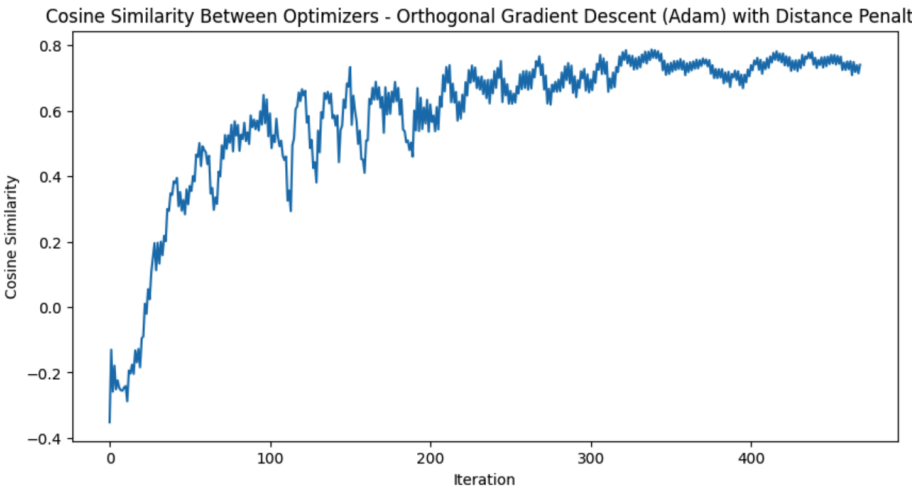
Analysis for Cosine Similarity Penalty:



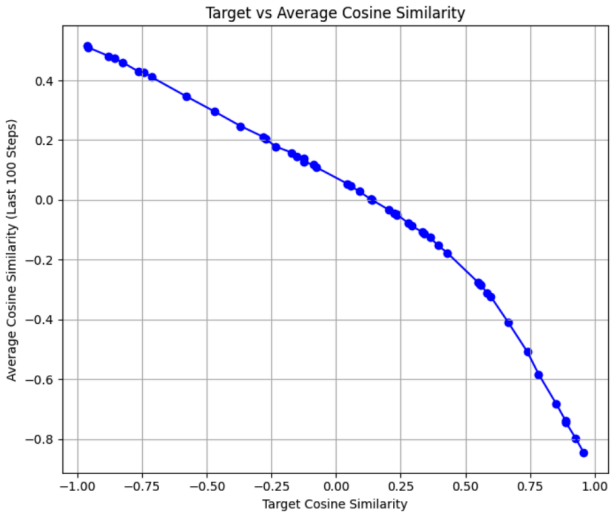


Average Loss: 0.2293
Average Cosine Similarity: -0.5141
Test Accuracy: 96.79%

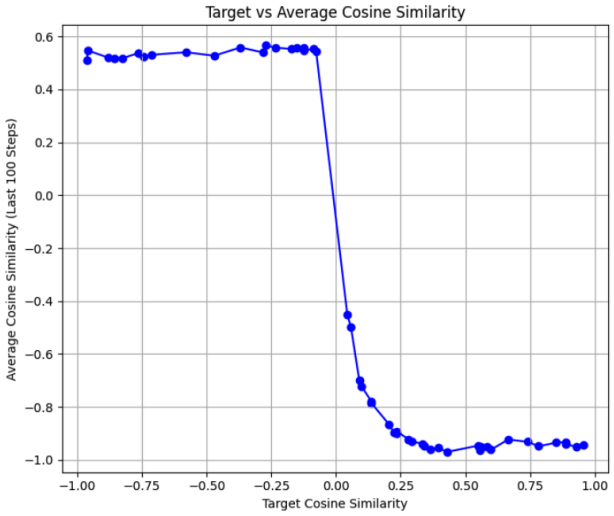




With the method I ran a test of 50 random target values
SGD optimizers



Adam Optimizers:



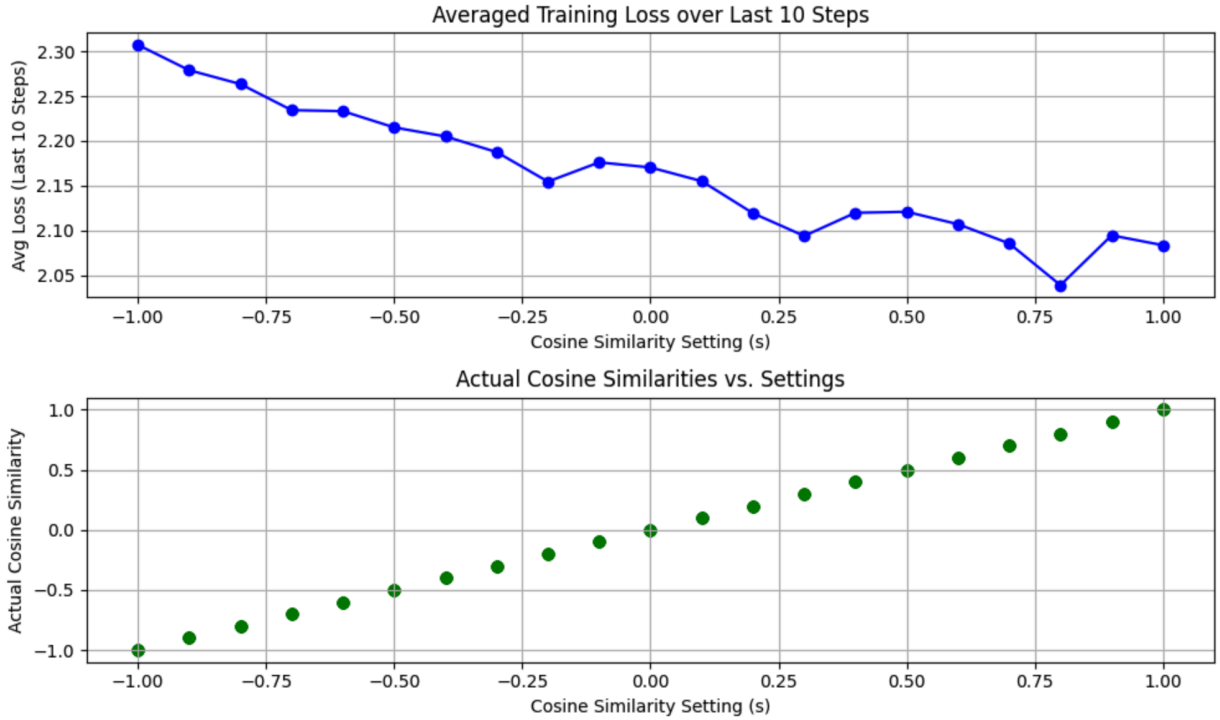
Implications: it depends on what values I am looking for.

- Any value around 0 I should use the SGD
- Any value high or low should use the adam optimizer

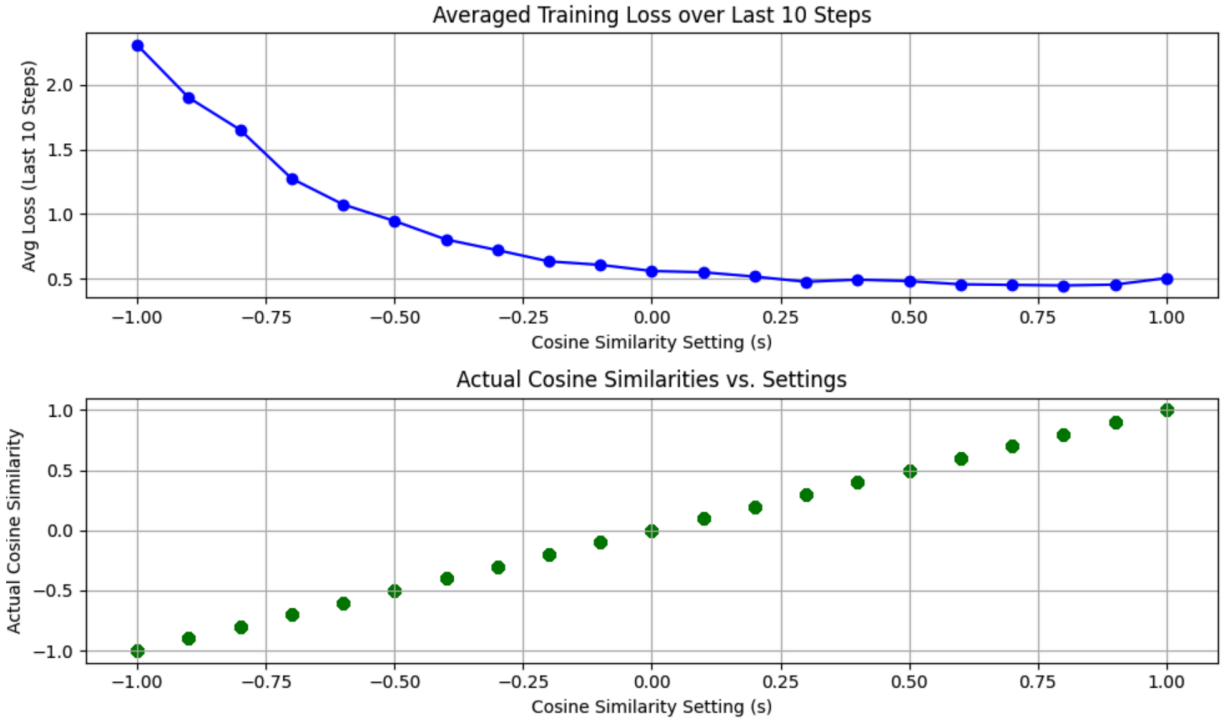
Adam in particular is interesting as it has a near -1 cosine similarity value but still very low training loss values

9-18 New method for selecting cosine similarity gives very good results. (100 steps per point, 1 epoch) SHUFFLE == FALSE, optimizers see the same data every time.

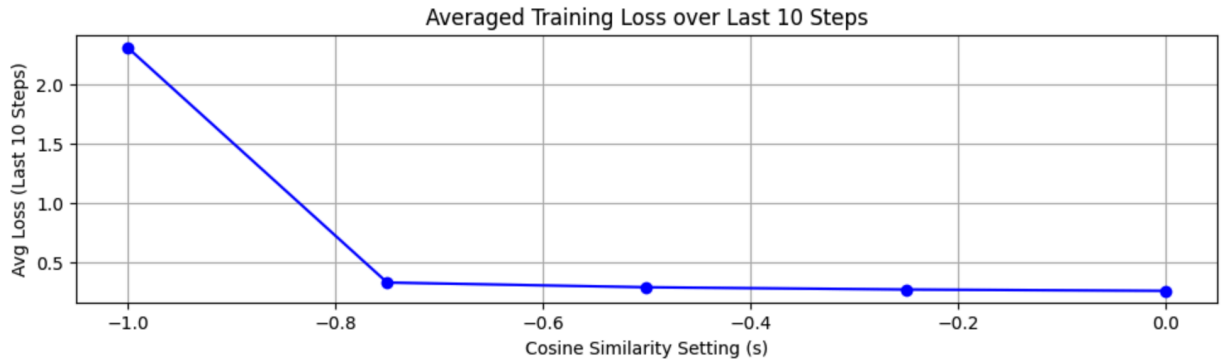
- Need to work on understanding more and confirming the results but they seem promising as a starting point.



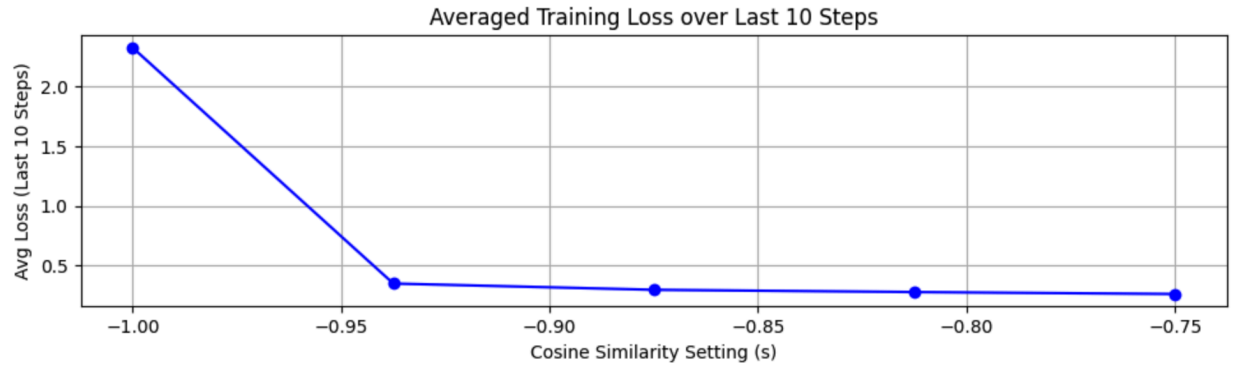
Repeat with 1000 steps per point, 1 epoch.



It also does still seem possible to train with higher - cosine similarity but it takes more time. (5 epochs, all data (no step limit)



Repeat but look closer at -1 -> -0.75 (10 epochs, no step limit)



So while it does take much longer to train I seem to be able to get extremely close to -1 cosine similarity in training.

Now I'll work on bringing these findings back to the memory loop work.

First step will be to get the cosine similarity in a standard training without messing with it for both adam and sgd.

I was looking at the memory loop method and it looks like the two optimizers are basically the exact same thing. Cosine similarity of almost perfectly 1 Which is why all my plots are dumb looking.

This makes a lot of sense since I am not controlling this at all.

Unusual though that I observed the hidden memory effect in the same location though? Your theory of multiple local memory could be wrong.

But then again some of the similarity values are over 1? I dunno

9-25-24: thoughts

It seems that the “true” solution to CL is almost impossible.

What if an approximation could be the following:

1. Train a model on a random mixture of many data
2. Freeze everything but a single component
3. Train the model on the specific task
4. Freeze the component unfreeze the system
5. Train on general data again
6. Next task/loop

Mapping these chunks of parameters via a mapped embedding of all the tasks?

Efficient computation:
- Take the vector input directly at the input layer of the parameters
- Finish inference and back prop
- if you put the parameter chunk near the end it would be pretty efficient

9-26-24

Adding to this more thoughts that one of the key problems of sequential memory is that memory is massively distributed over everything. If we could enforce structure on the model then we could perform something like the above far more effectively.

The issue is that the model itself has zero structure on how it gives us data and information.