FREEZE!: Pretrained Language Models as Frozen Feature Extractors for Semantic Tasks

Hayden McDonald
hayden_mcdonald@brown.edu

Nicholas Marsano
nicholas_marsano@brown.edu

Oliver McLaughlin
oliver_mclaughlin@brown.edu

Abstract

We investigate whether frozen language models can effectively serve as feature extractors for semantic tasks without any fine-tuning. By treating intermediate activations from different layers as independent sources of semantic information, we combine autoencoder pretraining with siamese networks to learn semantic representations. Using GPT2-medium as our base model, we achieve 76.86% Pearson correlation on the Semantic Text Similarity Benchmark (STSB) using a model containing less than 7% of GPT2-medium's parameter count. Our approach also produces an interpretable similarity measure for GPT2-medium's intermediate activations that aligns with human judgments of semantic similarity.

1 Introduction

Our project is an investigation into a few related areas. In short, we wanted to see if we could develop an architecture that could take the intermediate activations of a pre-trained decoder-only large language model (Henceforth LLM) and use them to do text-embedding tasks. We chose semantic text similarity (STS) as our primary focus and motivation because of its simplicity. This task involves training a model to rate the semantic similarity of a pair of sentences. In contrast to most other work using LLMs for text embedding, we are *not* fine-tuning the model, we are instead treating the model's intermediate activations as a frozen, *"feature extracted"* semantic representation.

We chose this idea for several reasons.

1. Our original motivational question was: *"Do 'similar' prompts (to human readers) have 'similar' (under some metric) representations to a given language model?"*. Without transforming the latent representations, similarity measures like cosine similarity do not correlate strongly with *human notions* of semantic similarity. This phenomenon is exemplified in Figure 1.
2. Small pretrained LLMs are cheap to do forward passes on and very plentiful. If we could extract useful text embeddings from their intermediate activations without fine-tuning the base model, we could get a lot of functionality for very little compute.
3. We also wanted to investigate the question: *Do multiple intermediate activations perform better than just the last?* Typically when doing finetuning or transfer learning you remove the classifier end and use the last intermediate representation as your "feature extracted representation". We wanted to see if there was any performance benefit in using *multiple layer's* worth of representation. This is clearly the case in the finetuning setting [16] but it was unclear to us if this would work in a frozen setting.
4. There just isn't that much work on this particular topic. For this task you'd typically finetune the LLM using LoRA [10] or simply just use an existing text embedding model. We

hoped to discover interesting properties and learn a lot about what it's like to try and *"disentangle"* an LLM's intermediate activations into something useful by pursuing this project. [1]

We chose `gpt2-medium` (355M parameters) as our *"base model"* for study because Oliver was familiar with its architecture and it readily fit onto all of our GPUs. We also did minor experiments on `qwen2-0.5B` (391M parameters). Moreover we chose `gpt2-medium` because of its very low quality text generation ability and verifiably low quality embeddings [8] (This paper tests the larger `gpt2` base model which is more powerful. Our model is a smaller, less performant version of that). It was not at all a priori obvious to us that its representations would be fit for this task.

1.1 Problem Setup

Figure 1 was our "starting point". This figure depicts the correlation between two important quantities for our investigation:

1. The cosine similarity of two sentences' latent representations (See 3.2 for more information on this representation)
2. A human rated similarity score for those same two sentences (See 3.1)

For each layer we computed the mean correlation between these two quantities for more than five thousand sentence pairs. This (low quality embeddings) is essentially the problem we are trying to solve – *How can we transform these latent representations into something useable for semantic text similarity?*

2 Related Work

Reusing pretrained LLMs for semantic tasks like STS is nothing new. There is extensive literature about finetuning and retraining these models for a variety of new tasks [14], [16]. Moreover using decoder-only models for embedding tasks is common, with works like SGPT [12], LLM2VEC [2], LLMEmbed [17] etc. exploring this idea as well as "meta-techniques" for improving these embeddings like Mitchell et. al's paper on how repeating a given piece of text within a prompt improves downstream embedding performance after collecting a truncated subset of the generated latents [15]. It is also well known that GPT-2 produces incredibly low quality embeddings [8] and even basic exploration shows that the smaller `gpt2-medium` produces very low quality text generation.

Our work also takes inspiration from Adapters [9] and other parameter-efficient fine-tuning methods. Adapters provide an alternative to full model fine-tuning by inserting small feed-forward networks between a pretrained model's layers and training only these small sub-networks. The key insight Adapters exploit is that modifying how information flows between layers can adapt the model to new tasks without changing its core weights. This approach preserves the model's original capabilities while enabling task-specific optimization.

3 Data

We have three major datasets that we used for both pretraining and supervised STS training.

1. STSB [7] – A standard sentence similarity benchmark with associated training and test sets. The sentence data is scraped from articles and news headlines and was rated by humans for their semantic similarity.
2. GenericsKB [3] – A 'generic sentence' dataset of which we used twenty-thousand examples. These sentences are varied in length and content and serve as our baseline distribution of sentence data.

[1] There was only one paper we could find doing a similar setup – extracting latents from an LLM and applying them to a task – but it is very low quality and does not reveal many of the important details [1]. There are many papers *like* this one ([11] comes to mind) but almost none completely freeze the entire model and "start from scratch" as we did here.

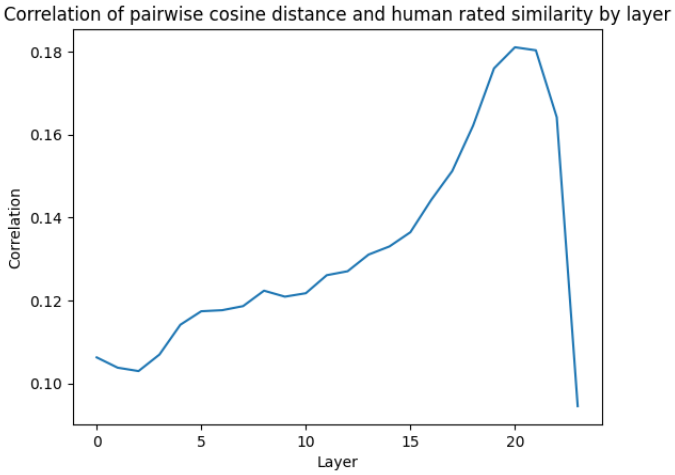

Figure 1: **The problem we are ultimately trying to solve**. Layerwise average correlation between gpt1-medium's embeddings and their cosine similarity vs. human similarity score for the STS training pairs

3. SNLI [5] – An inference relation dataset. Used in a pretraining experiment and in Sentence-BERT [14]. Most relevant to our discussion is that this dataset was used as a pretraining task in the Sentence-BERT paper [14].

3.1 Semantic Text Similarity Benchmark (STSB)

The main task we targeted was STSB [7], which is a benchmark for semantic text similarity. It's comprised of pairs of sentences and a human-rated "similarity score". For example:

$$(\text{s1='A plane is taking off.', s2='An air plane is taking off.', sim} = 1.0)$$

The performance metrics for STS are typically Spearman's rank correlation coefficient and Pearson correlation between the predicted similarities of pairs of sentences and the human rated similarities. State of the art for this task is `90 - 93%` in both spearman and pearson according to the MTEB leaderboard as of the writing of this document [13].

3.2 Data Collection and Pipeline

Our data pipeline is fairly simple. We're treating `gpt2-medium` as a feature extractor, so for each piece of text for any task, we simply pass that text through the model and extract a latent representation from the intermediate activations at each layer. We chose to take the *last token at each layer, after the residual connection* (Henceforth called a *latent*). This strategy is not original and is fairly standard in the *fine tuning* literature [17]. We expected it to do fine for our task given that `gpt2-medium` uses causal attention (and thus the last token has "seen" all of the previous tokens), even though we're not fine tuning. This gives us, for each sentence, twenty-four `1024` dimensional vectors (`n_layers x d_model`, `gpt2-medium` has twenty-four layers). One of the main challenges we wanted to overcome by taking this project on was finding a sensible way to *actually use* such a high volume and dimensionality of data.

We found no benefit from normalizing our activation data (I.e. center and then scale to unit variance) so we decided to not do so in our final model training. Rerunning our analysis confirms that external data normalization does not increase performance in our task (Likely due to the layer normalization employed by our autoencoders).

4 Methodology

The key insight behind our approach is that different layers in transformer models encode distinct types of information, with each layer potentially discarding or transforming information from pre-

vious layers as it builds towards its final objective. This idea is most clearly shown in Anthropic's work on Sparse Autoencoders for interpretability of LLMs [6] wherein different layers are shown to possibly be responsible for representing different and specific information.

Rather than relying solely on the final layers, we hypothesized that earlier layers might contain valuable semantic features that get discarded or transformed as information propagates through the model. This idea is most clearly shown in Figure 1, where the final layers of `gpt2-medium` perform worst for STS.

Our approach treats each layer's representational space as fully independent sources of semantic information. This is in contrast to methods like Adapters [9] which must maintain the original model's information flow. While Adapters can only modify how information passes between layers, our method can directly access and utilize the unique semantic information present at each layer.

4.1 Training and Loss function

Our general training scheme was the following:

1. Pass a given piece of text through `gpt2-medium`.
2. Extract the latents (Described in Section 3.2) from each layer. We take these latents and completely decouple them from the base model. In our actual experiments we pre-computed all of these latents and saved them to disk.
3. Pass those latents into our model.

To train our model to do STS we pass in both pairs of sentences to `gpt2-medium` as described above, pass those latent vectors into our model and extract a scalar similarity value. This allows us to do supervised similarity learning using the STSB dataset.

4.1.1 Loss function

Our loss function evolved throughout our experiments. Initially, we used mean squared-error (MSE) loss between the predicted similarities for a sentence pair and the human-rated similarities from the STSB dataset. For our final model, however, our loss function for supervised STS training departs from traditional MSE in that we only minimize the variance of the residuals, instead of both variance and bias which are traditionally minimized when using MSE.

Empirically we found that our residuals centered themselves near zero even though we didn't enforce this directly. We also found that taking the log of this value improved training stability and overall performance. We attribute this to the fact that the sample variance of a given set of residuals itself has high variance and thus the log term stabilizes gradients and overall training.

Our final loss function is then as follows:

$$\mathcal{L}(\theta) = \log(\text{Var}(y_\theta - y))$$

Where y_θ is the predicted similarity scores and y is the ground truth. We only saw meaningful improvement with this new loss metric on the final architecture, so our experimental results reported for all but the last model used MSE.

4.2 Architecture

Our architecture was decided through a series of experiments and trial and error. We tried to only make architectural decisions which were, in some way, *principled* – I.e. we had a decent reason for making that particular change.

4.2.1 Activation Choice

For every experiment detailed here we chose to use GELU as our activation function. Our rational for this is that our base model `gpt2-medium` uses GELU, so it's likely that its representation relies on the shape and quirks of that activation function. We did some experiments with other activation functions and found worse performance, suggesting that using other activation functions might waste parameters on "replicating" GELU-like transformations.

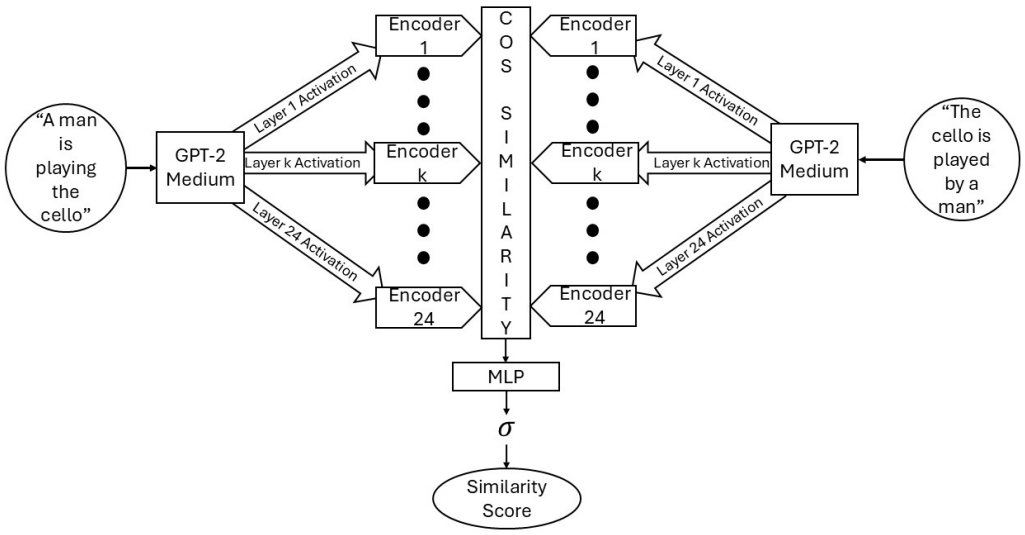

Figure 2: Final Model Diagram

4.2.2 First iteration

Our first architecture was incredibly simple – Just take the layerwise cosine similarities described in Section 1.1 and pass them through an MLP to do basic weighting and statistical correlation. We felt this was reasonable because although the individual correlations were quite weak, it wasn't clear to us that each layer was correlating *to the same content*. In the same way that many weak models can be combined to create a strong one, we figured that a MLP would be able to decipher some of the more complicated relationships, even just through the layerwise similarities.

For this first iteration we used a two layer MLP (We did not find better performance with deeper models) that took the twenty-four layerwise cosine similarities from `gpt2-medium`'s respective latents and outputted a sigmoided single scalar. This was trained using the scheme described in Section 4.1. This model significantly improved upon baseline, giving us a test set pearson correlation of `39.2%` and spearman of `42.06%` ($p << 0.001$)

4.2.3 Layerwise Siamese Networks

After the previous architectural experiment, we figured that we were probably bottlenecked by the representation of our base model, necessitating a transformation. Inspired by prior work on Siamese networks for semantic tasks [14], we decided to modify our architecture by applying siamese networks to each layer, computing the cosine similarities of those representations, and then finally passing those cosine similarities into a weighting/pattern matching MLP as described above. The intuition here is that each layer probably contains meaningfully different and useful information, as evidence by weighted cosine doing better than any particular layer. So, if we learn a nonlinear transformation of each layer's particular space through supervised STS training, we might be able to extract a better representation for this task and have overall better downstream performance.

This architecture achieved a significant performance increase, moving our pearson and spearman on the test set to `71.63%` and `69.72` ($p << 0.001$) respectively.

4.2.4 Autoencoder pretraining and the final model

We then realized that although this architecture was training quite well, it was likely the case that inputs which were significantly different from those in the input distribution would likely not be well represented by our siamese networks. This then inspired us to change our training strategy. Our current strategy coupled our representation learning and our supervised STS learning. That is, the model had to both learn a good representation of the input whilst also learning how to orient

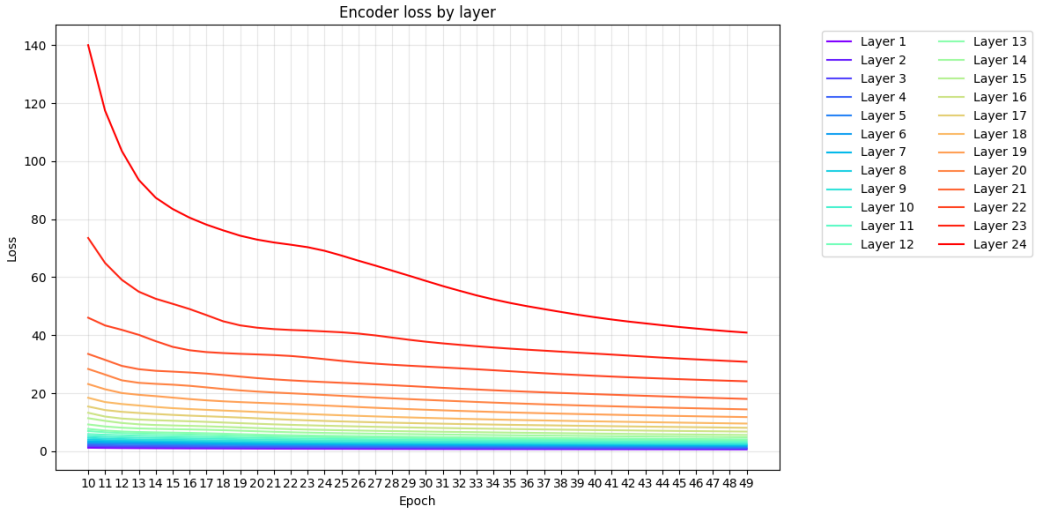

Figure 3: Autoencoder loss by layer. $d_{\text{bottleneck}} = 256$

examples spatially to reflect semantic similarity. We then decided to break this up by first training a set of layerwise autoencoders over twenty-thousand "generic" sentences from the GenericsKB [3] dataset. These sentences were similar in length and complexity to the STS training set that we felt it was an appropriate 'pretraining' dataset. We trained the autoencoders to reconstruct each layer's latent for each sentence (I.e. twenty-four autoencoders, each trying to reconstruct their respective layer's latent). We then did this pretraining and took the encoder part of each autoencoder and treated it as our new siamese network. Each autoencoder had a bottleneck of 256 hidden units.

Immediately, our results were *indistinguishable* from the results of the previous section. We then produced Figure 3 and realized that the layerwise losses for each autoencoder increased exponentially as the layers went on. This inspired us to, among many other things that did not work, widen the bottleneck of the last twelve autoencoders from 256 to 512. This then finally showed a performance increase to `75.36%` pearson and `73.21%` spearman. Our final change was to incorporate the loss function switch described in Section 4.1, which brought us to a final pearson and spearman correlation of:

$$\text{Pearson} = 76.86\%$$
$$\text{Spearman} = 75.03\% \ , \ p << 0.001$$

Moreover, this architecture actually ended up with more than $600,000$ fewer parameters when compared to our previous best model. Importantly, the *only* way we saw an increase in downstream performance, after many hyperparameter experiments and automated tuning, was to *widen the bottleneck at the later layers*. This heavily suggests that the information present at later latent vectors is not only much richer (And thus requiring many more bases to represent) but also incredibly important for semantic tasks. Critically, *deepening* the autoencoders *did not improve downstream performance*, even though doing so added millions of parameters. Only widening the bottleneck improved performance. Our model is depicted in Figure 2.

4.3 Experiments that failed

Alternate Latent Choice We tried several choices of latent including mean and sum pooled across all of the tokens at each layer. We found significantly worse performance from either approach. Intuitively this makes sense, only the last token has a "full picture" of the input and polluting that token with others would surely decrease performance.

Hyperparameter Optimization We ran several hyperparameter optimization/search runs on the autoencoders changing everything from their depth, dropout rate, width, etc. and found absolutely no benefit in downstream performance (Despite significant effort). We tried both manual approaches

and automated approaches using Optuna. Interestingly, we *did* see improvements in our autoencoder loss but did not see any downstream benefit – Suggesting our informational bottleneck is either the incoming latents or the architecture itself.

SNLI Pretraining Sentence-BERT [14] pretrained its siamese networks on the SNLI [5] relation task to great benefit. We tried this with our layerwise approach and found success in the SNLI task but no downstream benefits for STS.

Pooling and attention mechanisms We tried several schemes for pooling or otherwise combining both the latent representations coming from `gpt2-medium` and our learned representations. One pooling strategy we tried (Inspired by the 'semantic subspaces' idea of Bolukbasi et. al [4]) in the hopes of creating a single unified embedding was modifying the autoencoder training to instead of having twenty-four individual vectors to decode, we simply up-projected each encoder's bottlenecked output and summed accross each layer. This gave us a single high dimensional vector which we then fed into each "decoder", which was was trained to reconstruct the original layer's embedding from the unified one. The idea was that the unified representation would be encouraged to reuse subspaces to carry each layer's individual information so that it could be appropriately decoded. This model performed worse than our best model with a Pearson correlation of roughly `60%`. We do believe, though, that this scheme is probably the way forward if we want to expand to other tasks.

We also experimented with some of the attention and attention pooling strategies outlined in Tang et. al [16]. This did not do well when pooling either the original latents, nor the learned autoencoded representations.

Fake Data At one point we were wondering if we just simply didn't have enough examples of semantically similar sentences to break our best score, so we tried a few data generation strategies to see if we could mitigate this problem. Most notably we tried adding a little bit of noise to the activations to see if the model would generalize better – It didn't. We also tried modifying the prompts to change words or capitalization (To see if the difference in tokenization would change anything) – It didn't.

5 Results

5.1 Requirements for success

Our main notion of success was performance on the STSB [7] benchmark as described in Section 3.1. Pearson/spearman correlation are both reasonable measures of success on this task because they measure how consistent your model is with human-rated similarity notions, regardless of scale or center. State of the art on this task is `90-92%` for both Spearman and Pearson correlation as of the writing of this document [13]. Our goal was to achieve at least `60%` pearson correlation. We really did not have strong reasoning for why this might be a reasonable goal but preliminary results showed that at least `40%` was possible so we decided that `60%` was a reasonable stretch goal.

5.2 Final Results

We achieved a Pearson correlation on STSB's test set of `76.86%` [2] and a Spearman correlation of `75.3%` with a p-value of significantly less than `0.0001`. Ablation studies confirm both the utility of the unsupervised pretraining and the benefit of using multiple layers[3], in either case removing that step (or any layer) decreases performance. Our results are summarized in Figure 5. Our model's

[2] We were able to score up to 77.1% but were unable to consistently reproduce it. The score we report here is the one in our notebook.

[3] Doing meaningful ablation studies on our setup is difficult because our model is designed to train using multiple latents at a time. We tried just training a normal single-latent model and in every case performance suffered. That being said, to fully decide whether 'multiple layers are better than just one' would require investing significant time into tuning our setup to use a single latent. At least in this setup, we consistently found performance decreases when ablating any of the latents. Section 5.3.2 offers more insight into this question.

errors on the test set are quite favorably distributed, Figure 4 shows that the vast majority are centered near zero ($\mu = .06$, $\sigma = 0.197$) and the Q-Q plot shows that our model is distributed relatively symmetrically with thin tails, indicating no major over or underprediction. Contrary to our intuition, changing our loss function back to MSE did not bring the center of our residuals to closer to zero but instead just increased the variance.

We also performed some qualitative analysis on the highest error predictions (See Section 10 for more details) and note that although or model performs moderately well on the benchmark, it has clear and unavoidable limitations. Most notably, it has trouble distinguishing semantic and syntactic similarity and it did not learn to recognize identical sentences. Despite these limitations, we still consider our results a success because of our performance on STSB. Even though *some* of our model's predictions are quite bad, the vast majority of them are quite good, hence our STSB performance.

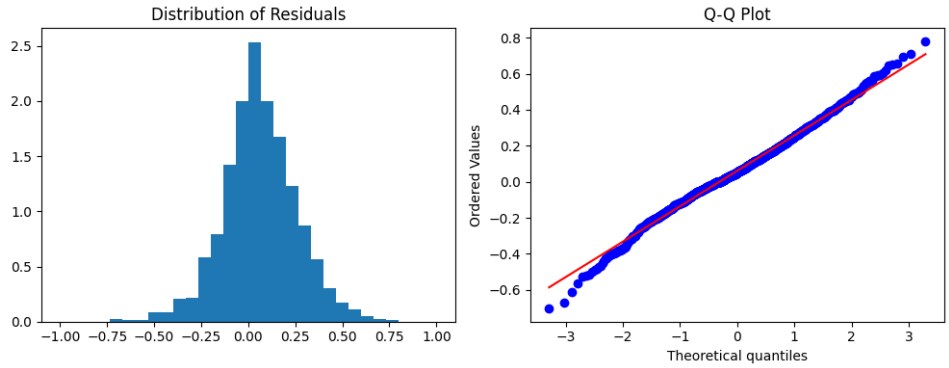

Figure 4: Density histogram and Q-Q plot of final model errors

5.3 Answering Our Original Questions

5.3.1 *Do 'similar' prompts have 'similar' representations?*

Our results suggest that, yes, under a particular transformation (our model), `gpt2-medium`'s latents do in fact correlate strongly with human notions of similarity. This implies that all of the necessary information to produce a decent sentence similarity model is encoded within the latent representations of the base model.

5.3.2 *Do multiple layers perform better than just the last one?*

This question was particularly interesting to us becuase if multiple layers did in fact perform better than just the final one, it would say quite a lot about the importance of information being lost during a forward pass.

In short, we found that using all layers works better than just picking one. Single-layer versions of our model (keeping the same architecture but only using one layer) performed notably worse, reaching only `71%` correlation. At first, this seemed like a meaningful comparison, but we later realized it's not particularly informative - our full model has 24 times more parameters than the single-layer version, and more importantly, if some layers weren't useful, the model could simply *learn to ignore them*. We now view the choice of which layers to include as just another hyperparameter to tune, rather than a fundamental architectural decision.

Interestingly, Figure 1 shows that the last latent in the residual stream was essentially *useless* for this task. This may suggest that previous evaluations of GPT-2 for embedding tasks, among others, may be fundamentally limited by their choice of latent.

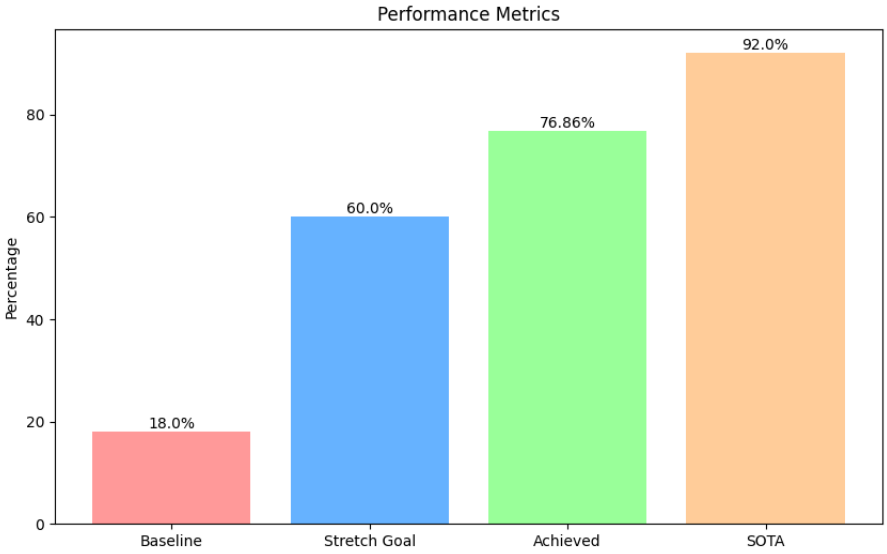

Figure 5: Final results – Pearson correlation

6 Ethics

6.1 Societal Impacts of our Approach

A critical ethical consideration in our work is that by using frozen LLM representations as our foundation, we inevitably inherit and potentially amplify any biases present in the base model. GPT-2-medium was trained on a broad internet corpus which likely contains various societal biases around gender, race, religion, and other sensitive topics. When we extract and transform its intermediate representations, we're not just capturing neutral semantic information - we're working with representations that may encode these problematic biases.

This issue connects to broader concerns about the responsibility of building on top of existing AI systems. While our approach is more computationally accessible than fine-tuning large models, this accessibility comes with the ethical burden of potentially propagating problematic biases to a wider range of applications. Future work should investigate methods for identifying and mitigating these inherited biases while preserving useful semantic information.

6.2 Why is Deep Learning a good approach?

Deep learning is particularly well-suited for our project because we're trying to make sense of LLM intermediate activations - complex, high-dimensional representations that are notoriously difficult to interpret by human analysis alone. Just as we use machine learning to find patterns in large datasets, we can use it to discover meaningful patterns in these internal model representations. Additionally, our approach demonstrates that we can extract useful capabilities from smaller models rather than always requiring larger, more computationally expensive ones. This pushes back against the trend of achieving better performance simply by scaling up model size and instead shows how targeted applications of smaller models can advance the field in meaningful ways.

However, there's an inherent irony in our approach: while attempting to create an interpretable measure of similarity for LLM activations, we've added another layer of complexity through our own neural networks. Though we succeeded in creating a useful similarity measure, we did so by essentially trading one black box for another. This highlights a fundamental tension in modern deep learning - sometimes making systems more capable can come at the cost of making them less interpretable. We expect this to become a fundamental problem in interpretability over the coming years. As models gain capability, we need a similar level of capability to understand them and with capability (in our current paradigm) comes further obfuscation.

7 Discussion

7.1 *"What did the model learn?"*

An interesting question to ask here is *What kind of semantic similarity did our model learn?* Although STSB has a particular idea of semantic similarity (Namely, that two sentences 'mean' the same thing), true *"semantic similarity"* is not limited to just identical ideas being expressed in different ways. We could easily define semantic similarity to be in regards to the abstract character of the sentences, or of the *ideas* they are conveying. For example, STSB labels the *"semantic similarity"* of the following sentences as 20%:

```
'Hawaii passes gay marriage bill', 'US Senate passes gay workers bill',
```

Our model labels them at 75.98%. Which is *"more"* correct? Although the literal content of the two sentences is different, they are quite similar in what they are conveying – Bills being passed regarding LGBT civil rights. There are numerous examples exactly like this one. One weaker example is:

```
'You will want to clean the area first.',
'You will also want to remove the seeds.',
```

Which STSB has rated at a similarity of 0%. Our model is less conservative in its estimate at 56.01%. Neither interpretation is wrong and clearly our model makes obvious and unavoidable errors (See Section 10 for more details) but in many contexts – such as following steps in a recipe – these sentences could be functionally identical, both indicating preparatory steps in a sequence.

This raises deeper questions about the nature of semantic similarity itself. Two humans, given different contexts or purposes (e.g., legal analysis vs. historical study), might rate these sentences quite differently. Our model's architecture, processing information across multiple transformer layers, may naturally capture these various levels of similarity – from surface-level linguistics to broader themes. Of course this does not distract from the fact that our model is nowhere near state of the art, but it is a worthwhile question to ask either way. Clearly our training produced *a particular interpretation of semantic similarity*, but which was it? Excluding clear errors, of course. We offer more insight into this in Section 10.

7.2 *"What did we actually create?"*

Although our original motivation was in the vein of interpretability, our approach combines several potentially useful components:

- A layerwise-independent[4] parameter-efficient approach to adapting existing LLMs to sentence similarity without fine-tuning
- A similarity measure for `gpt2-medium`'s activations that aligns with human judgments of semantic similarity, potentially offering more interpretable insights into the model's representations
- A representation learning scheme for LLM activations that preserves the base model's weights and information flow while enabling task-specific adaptation

While our performance does not match state-of-the-art models, these techniques might prove useful for understanding and utilizing frozen LLM representations in other contexts. Our technique is also quite parameter efficient, using less than seven percent of the base-model's total parameter count. In future work we aim to see how far these individual components can be improved.

7.3 *"What about a bigger base model?"* and the Representational Ceiling

We ran many, many hyperparameter experiments on the autoencoders and did not find any benefit (despite significant effort) in downstream performance beyond the aforementioned widening of the bottleneck at later layers. This suggests an important limitation of our setup – *The representational*

[4] As opposed to Adapters, which are layerwise *dependent*.

capability of our base model. In the finetuning scenario the entire "representational machinery" of the base model is appropriated for the new task. Millions of parameters are modified and retuned to throw away useless information (wrt the task at hand) and prioritize information that improves performance on the new selected metric. In our situation, we are *"stuck with"* whatever representation the original training for `gpt2-medium` produced. Critically, this representation is not tuned for semantic similarity tasks, but for next token prediction. It is likely that these two objectives require substantially different representations (and thus contextual information) of the input to perform well at.

A natural extension to our work would be to try a larger and more capable base model. Our initial experiments used `qwen2-0.5B`, which performed significantly better than `gpt2-medium` on both Pearson and Spearman correlation (+~2% on both). However, this gap disappeared once we made the changes to the loss function described in Section 4.1. After rerunning the experiments with both base models and the new loss function, we found no meaningful difference between the two despite `qwen2-0.5B`'s additional parameters. This likely suggests a limitation in our experimental setup for qwen2, given that we simply substituted the base model in our original architecture and only adjusted the input dimension of the autoencoders to match. A more thorough investigation of larger models would require architectural changes tailored to their specific parameter counts. We just wanted to assure that our approach was not uniquely suitable for `gpt2-medium`.

8 Reflection

We're very happy with how this project turned out. There were a couple of scares here and there (one horrible incident of accidentally swapping our train and test sets for a few days comes to mind) but overall the performance we reached was significantly past our goal and we learned an incredible amount. We're quite proud of how we were able to take such a weak model and turn it around into a, though not state of the art, mostly capable STS model.

At the start of this project we made a pact to try and be principled with each decision that came our way. For example, we made it a rule to never just make a model deeper or wider for no reason. We tried to stay as minimal as possible throughout the whole project and we feel that paid off. We believe that every decision in our architectural design process (Mostly described in Section 4.2) had good and clear motivation. We're also incredibly proud of our experimentation and overall scientific process.

If we had more time we would definitely choose to try and make our model a more general embedding model (Likely though some kind of pooling mechanism) and work towards applying it to the Massive Text Embedding Benchmark [13]. We have some other experiments that we briefly described in the section on architecture that we believe are the way forward in terms of overall performance, we just think they would take significant time and research effort to prove fruitful. Finally, we would do more thorough model comparisons (I.e. full-scale finetuning `gpt2-medium` to the task, using our architecture with different base models) so we have a clearer picture of how our architecture stacks up against other choices. There aren't many models on the MTEB or STSB leaderboards that are similar to ours, so it's hard to get perspective. Even parameter count isn't necessarily a good measure of how close we are because the vast majority of our "parameters" are not trained.

We were quite surprised none of the pooling and attention mechanisms worked significantly better than our architecture. Our key takeaway from those experiments is that the finetuning literature relies on being able to shape the entirety of the representation present in the base model. When you can't control every part of the representation (as in our scenario where we're only "reading" the activations) attention and other linear-combination based methods (Such as sum or mean pooling) just don't make sense. The base model likely learns to use different tokens in the residual stream quite differently, likely not in a way that can just be arbitrarily combined.

Our biggest takeway by far from doing this project is that research is an incredibly fun and collaborative process. We all thoroughly enjoyed all the exploration, reading, experimentation, and rebuilding our mental models over and over to try and make good decisions. As for what we learned, we all believe we made significant progress in our understanding and ability in, among other things, training and debugging neural networks, transformer models, metric and similarity learning, reading and consuming research papers, scientific writing, and collaborative research.

9 Division of Labor

- *Oliver.* Lead the research effort and designed and implemented the major architectural experiments. Took on most of the report writing.
- *Hayden.* Lead the analysis effort and discovered the bottleneck in the autoencoders. Performed the first manual hyperparameter experiments and investigated intermediate correlations that helped improve our understanding of the model. Worked with Nick to write the ethics section.
- *Nick.* Lead the automated hyperparameter optimization effort and proposed alternate approaches for improving our score. Performed a significant portion of the literature review and collected techniques and paths we could take. Worked with Hayden to write the ethics section.

Our group is quite happy with how the work was distributed and feel our individual strengths were able to be leveraged whilst also allowing each individual to specalize deeply into what they were interested in.

10 Appendix: Qualitative Analysis of Model Errors

To better understand our model's limitations, we sorted all sentence pairs in the STSB test set by the absolute difference between predicted and true similarity scores. Below are the five pairs with the largest errors, representing our model's most significant failures.

```
"Not 'hiding the ball' on Russia: Obama",
"Obama says he's not 'hiding the ball' on Russia",
True similarity: 0.8800
Predicted: 0.2084

"Taiwan's economy grows 2.27% in April-June quarter",
"China's economic growth rebounds to 7.8% in latest quarter",
True similarity 0.2000
0.8955

'A man is laughing with a woman',
'A man and a woman laughing.',
True similarity: 0.9600
Predicted: 0.2572

'Supreme Court to Hear Voting Rights Act Case',
'Supreme Court to hear corporate human rights case',
True similarity: 0.2800
Predicted: 0.9893

"Hassan Rouhani wins Iran's presidential election",
'Maduro wins Venezuelan presidential vote',
True similarity: 0.0400
Predicted: 0.8193
```

These are just the worst performing sentence pairs but they show a trend seen by browsing the top fifty or so. Clearly our model has a different idea of similarity than STSB does, but that model isn't always completely off base. Consider the last sentence pair in this set. Both sentences describe a particular figure winning a particular election. Obviously, semantically, these two sentences are nothing alike, they are describing two completely different figures winning two completely different elections, but they are "semantically similar" in a more abstract sense. Similarly with the Taiwan/China pair and the supreme court pair. Did our model learn to recognize these as similar? Or is it just picking up surface level syntactic similarity? Regardless, the model clearly also just fails to see the similarity in some pairs, as seen in the man and woman laughing pair. Interestingly, the model also produces the following prediction:

```
'A brown dog is jumping.',
'A brown dog is jumping',
True similarity: 1
Predicted: 0.4735
```

Which shows that the model has *not learned to recognize completely identical sentences*. This might imply that our training set could use more identical pairs or that our model needs to be restructured in such a way that the ones vector (`(1, 1, ..., 1)`) produces a `1` at the end of the MLP. This is because our layerwise siamese networks use cosine similarity, so the output at each layer for two identical prompts is always the ones vector. In any case, this is a significant limitation of our architecture. Moreover many of these examples show how our model confuses surface level syntactic similarity with deeper semantic similarity.

References

[1] Mahmoud Basharat and Marwan Omar. Harnessing gpt-2 for feature extraction in malware detection: A novel approach to cybersecurity, 2024.

[2] Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. Llm2vec: Large language models are secretly powerful text encoders. (arXiv:2404.05961), August 2024. arXiv:2404.05961 [cs].

[3] Sumithra Bhakthavatsalam, Chloe Anastasiades, and Peter Clark. Genericskb: A knowledge base of generic statements, 2020.

[4] Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings, 2016.

[5] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[6] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[7] Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation. *CoRR*, abs/1708.00055, 2017.

[8] Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings, 2019.

[9] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

[10] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[11] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines, 2021.

[12] Niklas Muennighoff. Sgpt: Gpt sentence embeddings for semantic search, 2022.

[13] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022.

[14] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[15] Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. Repetition improves language model embeddings. (arXiv:2402.15449), February 2024. arXiv:2402.15449.

[16] Yixuan Tang and Yi Yang. Pooling and attention: What are effective designs for llm-based embedding models?, 2024.

[17] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. (arXiv:2401.00368), May 2024. arXiv:2401.00368.