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

December 5, 2024

Contents

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 sentence 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. Importantly, in contrast to most other work using LLMs for text embedding, we are *not* fine-tuning the model. We are simply taking a subset of the intermediate activations and treating them as a kind of *"feature extracted"* (frozen) 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?"*. By training a sentence-similarity model on LLM activations, we believed we could make progress on answering this question by *learning* a transformation of the activations that correlates to semantic similarity.

2. *If* we could train a small model to interpret a given LLM's activations and produce a useful text embedding (or set of embeddings) from them, we could skip costly fine-tuning and simply pre-train and fine-tune the small "interpreter" model. Pretrained small LLMs are cheap to do forward passes on and very plentiful, so if we could *just* use the intermediate activations and not have to do backward passes 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 [11] but it's unclear if it will work here.

4. From an interpretability standpoint, most distance or similarity measures (E.g cosine similarity) do not correlate strongly with *human ideas* of similarity (Figure 1). By training a similarity learner on LLM activations, grounded in human ideas of similarity, we hoped to produce a more interpretable measure of similarity for LLM activations.

5. There just isn't that much work on this particular topic. For this task you'd typically finetune the LLM using LoRA [6] 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're not really interested in *what* information is where, we just want to see *if it's useable* for a simple task like STS.

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 [**?**] (This paper tests the larger `gpt2` base model which is more powerful. Our model is a smaller, less performance 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 is essentially the problem we are trying to solve – *How can we transform these latent representations into something useable for semantic text similarity?*

In short, we're trying to solve the problem of "low quality" embeddings in small LLMs like `gpt2-medium` by finding an architecture which can leverage the information contained within its latents. Our target task for this experiment is semantic text similarity and we will measure success through our performance on the STSB [5] benchmark. To do this we trained a similarity learner on the STS dataset with some added unsupervised pretraining. We believe that if we can take these latent representations generated by `gpt2-medium` and train a model to effectively use them that this would be evidence of these embeddings being much higher quality than originally expected in terms of their information content.

[1] There was only one paper we could find doing a similar setup – extracing 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 ([7] comes to mind) but almost none completely freeze the entire model and "start from scratch" as we did here.

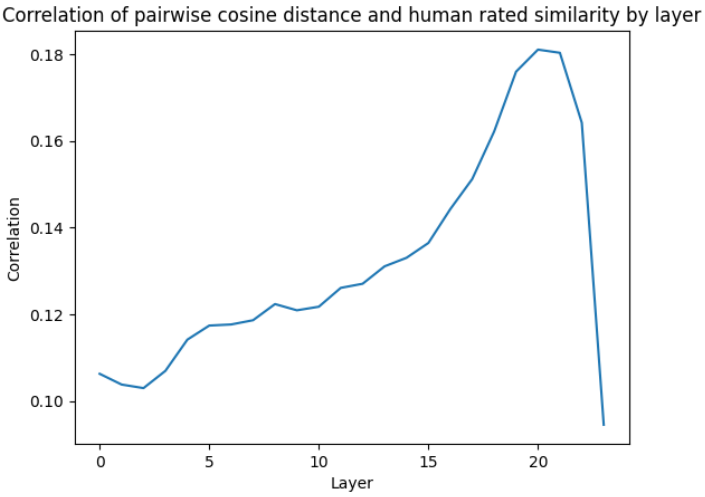

Figure 1: **The problem we are ultimately trying to solve**. Layerwise average correlation between gpt1-medium's embeddings and their cosine similarity vs. human similarity score for the STS training pairs

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 [9], [11]. Moreover using decoder-only models for embedding tasks is common, with works like SGPT [**?**], LLM2VEC [2], LLMEmbed [12] 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 benchmark performance after collecting a truncated subset of the generated latents [10]. It is also well known that GPT-2 produces incredibly low quality embeddings [**?**] and even basic exploration shows that the smaller `gpt2-medium` produces very low quality text generation.

3 Data

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

1. STSB [5] – A standard sentence similarity benchmark with associated training and test sets.
2. GenericsKB [3] – A 'generic sentence' dataset of which we used twenty-thousand examples.
3. SNLI [4] – An inference relation dataset. Used in a pretraining experiment and in Sentence-BERT [9]

3.1 Semantic Text Similarity Benchmark (STSB)

The main task we targeted was STSB [5], which is a benchmark for semantic text similarity. It's comprised of pairs of sentences and a human-rated "similarity score". For example:

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

The performance metrics for STS are typically Spearman's rank correlation coefficient and normal Pearson correlation between the predicted similarities of pairs of sentences and the human rated similarities. State of the art for this task is around `90 - 92%` in both spearman and pearson according to the MTEB leaderboard [8].

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 [12]. 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`). 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.

4 Methodology

4.1 Training and Loss function

Our general training scheme was the following:

1. Pass whatever example sentence for the current task 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. They are effectively just a set of vector representations of the input. No gradient is passed back through `gpt2-medium`
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. We then do supervised training where the ground truth is the human-rated similarity value of the two sentences as given by 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. While our model's performance is evaluated using both Pearson and Spearman correlation (which measure linear and monotonic relationships respectively), we found that simply minimizing residual variance without enforcing zero bias led to strong performance on both metrics. Since our model uses a sigmoid activation at the output layer, both our predictions and ground truth are bounded in [0,1], which naturally constrains the residuals to [-1,1]. This architectural choice means the residuals cannot have arbitrary bias even though we don't explicitly optimize for it. Given these bounds, we can focus our training objective purely on minimizing variance to promote consistent relationships between predictions and ground truth. Empirically we found that our residuals centered themselves near zero even though we didn't enforce this directly. We found that taking the log of this value improved training stability and overall performance. The log transformation provides more balanced gradients across different scales of variance, preventing optimization instability when variances are very large while maintaining sensitivity to improvements when variance is already small. 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 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 simple MLP would be able to decipher some of the more complicated relationships, even just through the layerwise similarities. This model was quite simple, just a one layer mlp 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%` ($p << 0.001$)

4.2.2 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 [9], 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

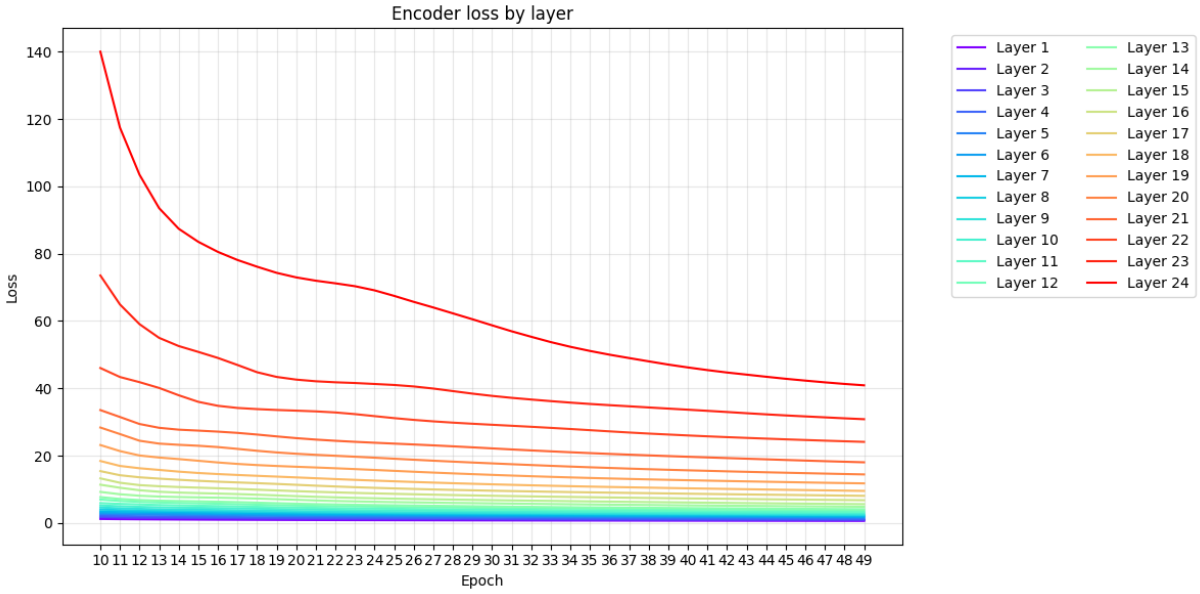

Figure 2: Autoencoder loss by layer. $d_{\text{bottleneck}} = 256$

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.

4.2.3 Autoencoder pretraining and the final model

We then realized that although this architecture was traning 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 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 senteces 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 2 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 incorperate 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 paramters. Only widening the bottleneck improved performance.

4.3 Experiments that failed

5 Results

5.1 Requirements for success

Our main notion of success was performance on the STSB [5] 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 [8]. 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%` and a Spearman correlation of `75.3%` with a p-value of significantly less than `0.0001`.

6 Ethics

7 Discussion

7.1 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 aformentioned widening of the bottleneck at later layers. This suggests an important limitation of our setup – *The representational 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 represenations (and thus contextual information) of the input to perform well at.

7.2 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 `0.0`. 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. This observation, supported by our finding that wider bottlenecks in later layers improved performance, suggests that our model's "errors" might actually be revealing limitations in how we define and measure semantic similarity, rather than just limitations in the model itself. 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.

8 Reflection

One big takeaway from this project is that as gigantic open source LLMs become more pevalent, it might be possible to chop off most of their layers and train little "interpreter" modules as we did here to leverage the incredibly rich and complex learned represenations at a fraction of the cost and computation time.

9 Division of Labor

- *Hayden.*
- *Nick.*
- *Oliver.*

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. That being said, 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.

11 Appendix: *STSB* dataset leakage

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] 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.

[5] 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.

[6] 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.

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

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

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

[10] 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.

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

[12] 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.