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 be used effectively as feature extractors for semantic tasks without any fine-tuning. Our approach takes the intermediate activations of GPT2-medium and treats them as independent sources of semantic information, processing them through a combination of autoencoder pretraining and siamese networks. Despite GPT2-medium's known limitations in generating quality embeddings, we achieve `76.86%` Pearson correlation on the STSB benchmark without touching the base model's weights. Interestingly, we found that widening the bottleneck of our autoencoders at later layers significantly improved performance, suggesting these layers encode richer semantic information that requires more representational capacity to capture. While not state-of-the-art, our results demonstrate that frozen LLM representations contain substantial semantic information that can be extracted efficiently, offering a possible lightweight alternative to traditional fine-tuning approaches.

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 [12] 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 [7] 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.

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 [5] (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 [4] 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.

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 [10], [12]. Moreover using decoder-only models for embedding tasks is common, with works like SGPT [8], LLM2VEC [1], LLMEmbed [13] 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 [11]. It is also well known that GPT-2 produces incredibly low quality embeddings [5] and even basic exploration shows that the smaller `gpt2-medium` produces very low quality text generation. Our work also takes inspiration from other parameter-efficient finetuning methods such as Adapters [6].

3 Data

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

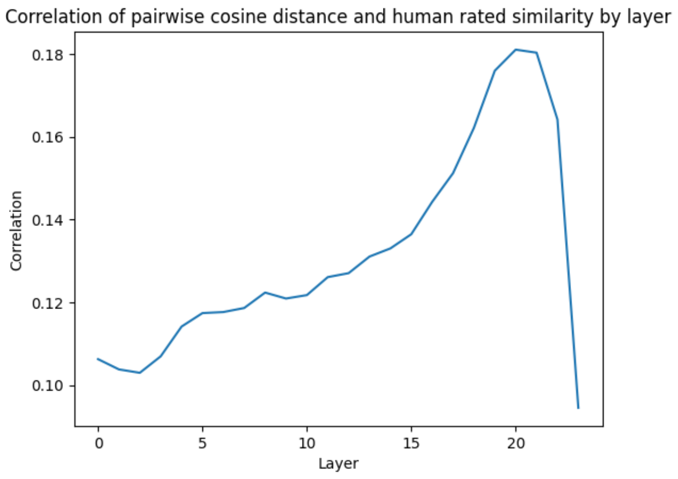

Figure 1: **The problem we are ultimately trying to solve**. Layerwise average correlation between gpt1-medium's embeddings and their cosine similarity vs. human similarity score for the STS training pairs

1. STSB [4] – A standard sentence similarity benchmark with associated training and test sets.
2. GenericsKB [2] – A 'generic sentence' dataset of which we used twenty-thousand examples.
3. SNLI [3] – An inference relation dataset. Used in a pretraining experiment and in Sentence-BERT [10]

3.1 Semantic Text Similarity Benchmark (STSB)

The main task we targeted was STSB [4], 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 [9].

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 [13]. 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.

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 previous layers as it builds towards its final objective. Rather than assuming later layers contain all necessary information, we hypothesized that earlier layers might preserve valuable semantic features that get filtered out during the progression through the model. While this is similar in concept to adapter models (CITE ME), our approach has a key advantage: since we're not constrained by the need to maintain the original model's information flow, we can treat each layer's representational space as fully independent. This allows us to potentially capture and utilize semantic features that might otherwise be lost or transformed as information propagates through the model's layers.

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 [10], 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.

4.2.3 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 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 [2] 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 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 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.

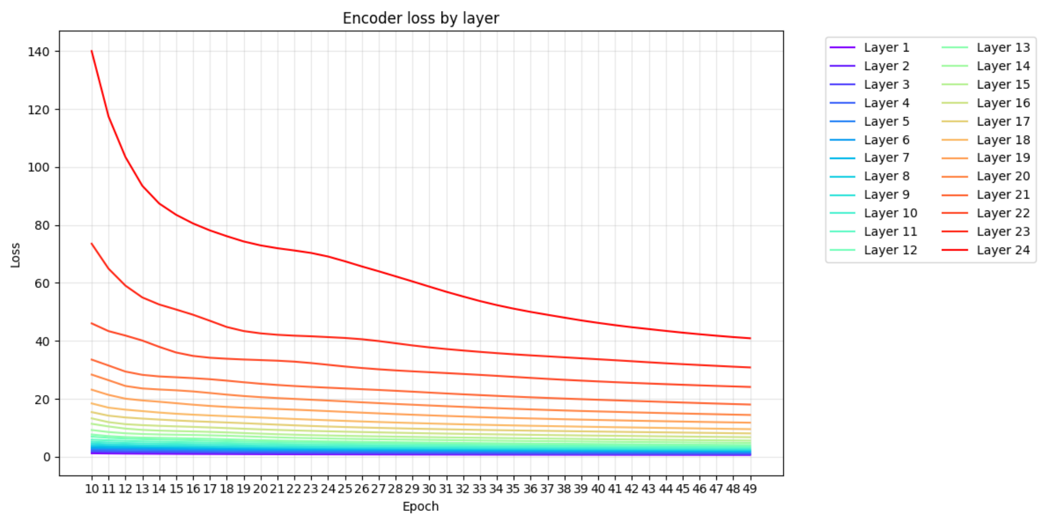

Figure 2: Autoencoder loss by layer. $d_{\text{bottleneck}} = 256$

4.3 Experiments that failed

Latent choice We tried several choices of latent including mean and sum pooled accross 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 [10] pretrained its siamese networks on the SNLI [3] 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 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 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 representation 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 [12]. This did not do well when pooling either the original latents, nor the learned autoencoded representations.

5 Results

5.1 Requirements for success

Our main notion of success was performance on the STSB [4] 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 [9]. 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`. Ablation studies confirm the utility of the unsupervised pretraining.

6 Ethics

There are two main societal impacts of this project. First, it addresses the relatively high barrier of entry for STS tasks by showing promise in lightweight methods that bypass the need for resource-intensive fine-tuning. This bypass offers a glimpse into the democratization of advanced NLP capabilities for researchers or organizations with limited computational resources. As more LLMs become open source, the barrier of entry to leverage them in practical applications can be lowered and allow for more innovation in smaller-scale environments. Second, the project addresses the advancement of model interpretability to increase trust in deployed models. By training an architecture to derive meaningful embeddings from intermediate activations, it contributes to the broader understanding of how LLMs encode semantic information. By helping to bridge the gap between machine representation and human comprehension, this project helps to further AI transparency as it plays a more dominant role in everyday life.

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 distinct, layerwise-independent 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. 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 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`.

7.4 *"Isn't this just an adapter?"*

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 [9]. 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.

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

- *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. Moreover many of these examples show how our model confuses surface level syntactic similarity with deeper semantic similarity.

11 Appendix: *STSB* dataset leakage

References

[1] 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].

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

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

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

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

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

[11] 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.

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

[13] 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.