NEURAL EMBEDDINGS FOR TEXT

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a new kind of embedding for natural language text that deeply represents semantic meaning. Standard text embeddings use the vector output of a pretrained language model. In our method, we let a language model learn from the text and then literally pick its brain, taking the actual weights of the model's neurons to generate a vector. We call this representation of the text a neural embedding. With analysis of its behavior on several datasets, we confirm the ability of this representation to reflect semantics of the text. We also compare neural embeddings with GPT sentence (SGPT) embeddings. We observe that neural embeddings achieve comparable performance with a far smaller model, including state of the art for several embedding benchmarks.

1 INTRODUCTION

Capturing the semantic meaning of text as a vector is a fundamental challenge for natural language processing (NLP) and an area of active research (Giorgi et al., 2021; Zhang et al., 2020; Gao et al., 2021; Huang et al., 2021; Yan et al., 2021; Zhang et al., 2021; Muennighoff, 2022; Alexander Liu, 2022; Chuang et al., 2022). Recent work has focused on fine-tuning pretrained language models with contrastive learning, either supervised (e.g. Reimers & Gurevych (2019); Zhang et al. (2021); Yan et al. (2021)) or unsupervised (e.g. Giorgi et al. (2021); Gao et al. (2021)). The embedding is generated by pooling the outputs of certain layers of the model as it processes a text.

Motivated by the need for deeper semantic representations of text, we propose a different kind of embedding. We allow a language model to fine-tune on a text input, and then measure the resulting changes to the model's own neuronal weights as a *neural embedding*. Because the changes in a model's neurons as it learns from text are practically linked to its language skills, such embeddings may reflect deeper semantic features of the text. If neural embeddings even just capture features of text that are orthogonal to those of standard embeddings, they could be a valuable tool for NLP tasks such as semantic search, classification, and text clustering.

We demonstrate that neural embeddings do indeed represent the semantic differences between samples of text. We evaluate neural embeddings on several datasets and find them comparable to the state of the art GPT sentence (SGPT) embeddings[1] (Muennighoff, 2022). But we also show that neural embeddings behave differently (Section 3). We achieve this with a far smaller underlying model: BERT-base-uncased, with 110M parameters, rather than the 5.8B-parameter model used by SGPT. Our contribution:

1. We propose a new kind of text representation: *neural embeddings*[2] (Section 2).
2. We evaluate embeddings by using several datasets and several criteria (Section 3). We show that by these criteria the neural embeddings are comparable to SGPT embeddings, that the errors they do are often different, and that they represent the qualities of the text differently.

[1] https://github.com/Muennighoff/sgpt
[2] github url will be provided here. Code is in supplementary material.

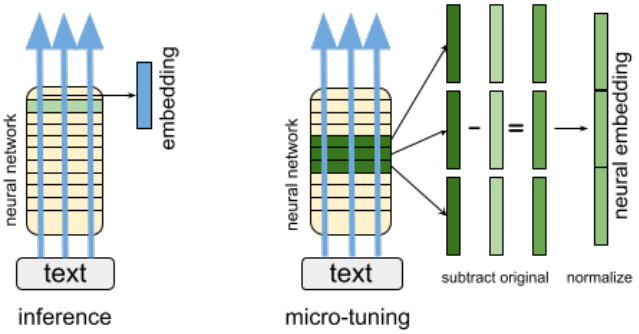

Figure 1: Illustration comparing the usual output embeddings (left) and neural embeddings (right). The output embeddings are taken from aggregated outputs of certain layers at inference, as shown on the left. Neural embeddings are taken from weights of certain layers at micro-tuning.

2 NEURAL EMBEDDING METHOD

The technique for generating neural embeddings is inspired by *micro-tuning*, first introduced for the BLANC-tune method of document summary quality evaluation[3] Vasilyev et al. (2020). It is a tuning on one sample only, and the tuned model is used for the sample only.

For each text sample, we start with the original language model and fine-tune only a few selected layers $L_1, L_2, ..., L_m$ while keeping all other layers frozen. Once the fine-tuning on the text sample is complete, we measure the difference between the new weights W'_j and the original weights W_j of each layer L_j and normalize the resulting vector. We obtain the neural embedding of the text by concatenating the normalized vectors:

$$E = \frac{E_c}{|E_c|}\ , \quad E_c = \|_{j=1}^{m}(W'_j - W_j)/|W'_j - W_j| \tag{1}$$

Schematically, our illustration is in Figure 1. For clarity, the algorithm is shown in Figure 2.

Given:
  Model M
  Weights $\{W_j\}$ from selected layers $\{L_j\}$ of M, all other layers are always frozen
Obtain embedding for text T:
Split $T = T_0 + T_1 + ...$
  Each text chunk T_i consists of max number of sentences fitting input size of the model M
Set model layers $\{L_j\}$ weights to $\{W_j\}$
Initialize $inputs_and_labels$
for each chunk T_i:
  Make inputs and labels
  Add inputs and labels to $inputs_and_labels$
Tune model M on $inputs_and_labels$
Take the tuned weights $\{W'_j\}$ of the layers $\{L_j\}$.
Calculate changes with respect to original weights: $dW_j = W'_j - W_j$
Normalize and concatenate them: $E_c = \|_{j=1}^{m} dW'_j/|dW_j|$
Normalize the result: $E = E_c/|E_c|$
return E

Figure 2: Producing neural embedding for text T. Notice that the weights of layers $\{L_j\}$ are always reset to the original values before obtaining neural embedding for a text.

[3]https://github.com/PrimerAI/blanc

For example, if we select three layers from the standard BERT base model, and if each selected layer has 768 weights, then the resulting embedding will have size $768 * 3 = 2304$. Before entering Equation 1, the weights of each layer are flattened from their possibly multidimensional tensor form.

Through this paper we use the pretrained transformer Devlin et al. (2019) model *bert-base-uncased* from the transformers python library Wolf et al. (2020). We found that layers either from the top of the model, or from the last transformer block of the model perform best. In the next section we use the following selection:

1. $L_1 = cls.predictions.transform.dense.bias$
2. $L_2 = cls.predictions.transform.LayerNorm.weight$
3. $L_3 = cls.predictions.transform.LayerNorm.bias$

During micro-tuning, the model is tasked with its pretraining objective. In the case of BERT, that is the masked token task. We chose a periodic masking strategy, wherein the masking (or absence of masking) repeats at every Pth token. Consider the masking blueprint (k, m). To obtain an input from the text, we keep the first k tokens of the text and mask next m tokens, repeating this pattern to the end of the text. We then generate our second input by shifting the pattern by 1 token, followed by shifting 2 tokens, and so on up to $k + m - 1$ tokens. Moreover, we can create inputs from not one but several blueprints: $[(k_1, m_1), ..., (k_n, m_n)]$. This algorithm for creating inputs is shown in Figure 3.

```
Given:
    Blueprints for masking: [(k_1, m_1), ..., (k_n, m_n)]
    Text chunk tokens T = [t_0, t_1, ...]
Initialize empty list inputs_and_labels
for each masking blueprint (k_i, m_i):
    Period P = k_i + m_i
    Maximal shift S_max = min(P, len(T))
    for s in range(S_max):
        input I = deepcopy(T)
        initialize labels L, with length len(L) = len(T)
        for j in range(len(T)):
            offset R = (j - s)%P
            if R >= k_i:
                set label: L[j] = I[j]
                mask token: I[j] = mask
        Append duple (I, L) to inputs_and_labels
return inputs_and_labels
```

Figure 3: Creating inputs and labels from a chunk of text. The given length of the chunk $len(T)$ must not exceed the max allowed size of model input.

All the inputs, randomly shuffled, constitute a 'dataset' for the micro-tuning. We found especially useful to have a mix of the simplest masking blueprints: $[(2, 1), (1, 1), (1, 2), (1, 3)]$. Figure 4 is an illustration for obtaining the inputs for text of 6 tokens with the blueprints $[(2, 1), (1, 1), (1, 2), (1, 3)]$. There are only 12 inputs for any text fitting into model maximal input size - or even less than 12 inputs if the text length is less than 4 tokens. This fits into single batch. In our evaluations (Section 3) we use 10 epochs micro-tuning with learning rate 0.01. See Appendix A about the processing time and the factors to reduce it.

Interestingly, if we do not mask tokens at all (but require prediction for all tokens), we still get reasonable embeddings, albeit with much lower 'quality' if assessed as we do in Section 3. The loss is still defined by cross entropy, no matter how easy the prediction task is, and for the same reason there is always weights difference in Equation 1.

3 EVALUATIONS

The goal of this section is to demonstrate that neural embeddings do capture the semantic differences between texts, and that on average the texts with similar meanings get close to each other in the

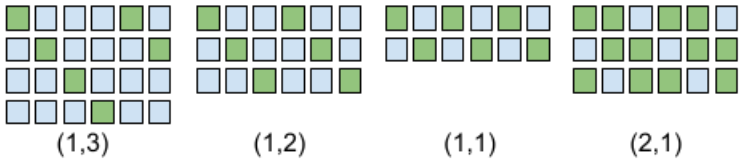

Figure 4: The inputs for micro-tuning on text with length 6 tokens, obtained by the blueprints $[(2,1),(1,1),(1,2),(1,3)]$. The tokens kept (not masked) are colored green. Labels are made for predicting the masked tokens. Altogether there are 12 inputs, represented here by 12 rows of length 6.

neural embedding space. No embedding can be ideal for all purposes; here we attempt to observe the behavior of neural embeddings on different data and from different points of view.

3.1 Triplets anchor-positive-negative

If we compare the semantic similarity between texts B and C with regard to text A, and if B is semantically closer to A, then we would want the scalar product of the embeddings of B and A to be higher than the product of the embeddings of C and A:

$$sim(A,B) > sim(A,C) \iff (E_A * E_B) > (E_A * E_C) \tag{2}$$

We will take triplets of texts with the similarity sim satisfying $sim(A,B) > sim(A,C)$ from several datasets, and we will count the fraction of triplets for which Equation 2 is not satisfied.

We can get the triplets (A,B,C) from any dataset of texts grouped by semantic similarity, so that semantically similar texts belong to the same group. Such selection of triplets was used for grouped images in Wang et al. (2014); Hoffer & Ailon (2015) and in related computer vision works. Selecting 'anchor' A and 'positive' B from the same group, and 'negative' C from any other group provides a test: We should verify $(E_A * E_B) > (E_A * E_C)$. With a dataset in hand, we create all possible triplets and then count the fraction of the triplets for which this condition is not satisfied.

The datasets we use, listed with the short notations of Table 1:

1. *mrpc*: Composed of all sentence pairs that were annotated as similar in the 'train' dataset of MRPC - Microsoft Research Paraphrase Corpus Dolan & Brockett (2005)[4]. There are 2474 such pairs, hence we have 2474 groups, each group consisting of two texts (sentences).
2. *sts*: Composed of all sentence pairs that were annotated with similarity score at least 4 (on the scale 1-5) in STS 'test' (Englsh language) dataset (Cer et al., 2017; May, 2021)[5]. There are 338 such pairs, hence we jave 338 group, each group consisting of two texts (sentences).
3. *catasaurus*: First 5000 pairs of paraphrases from the dataset catasaurus_paraphrase_dataset2[6]
4. *asset/t*: Subset 'test' of the dataset asset[7] Alva-Manchego et al. (2020) - the dataset of simplified sentences.
5. *cestwc/t*: 5000 samples from the subset 'tapaco' of the dataset cestwc/paraphrase[8] dataset. The samples are selected as first 500 classes (by labels), with 10 first texts selected for each class.
6. *cestwc/q*: Composed as *cestwc/t*, but using the subset 'quora'.
7. *cestwc/c*: Composed as *cestwc/t*, but using the subset 'coco'.
8. *summ-g*: 1700 generated summaries of the dataset SummEval[9] Fabbri et al. (2021). Each summary is labeled by the document for which it was generated; there are 100 documents, 17 generated summaries for each document.
9. *summ-r*: 1100 human-written summaries of the dataset SummEval. Each summary is labeled by the document for which it was generated; there are 11 reference summaries for each document.

[4]https://huggingface.co/datasets/glue/viewer/mrpc/train
[5]https://huggingface.co/datasets/stsb_multi_mt
[6]https://huggingface.co/datasets/catasaurus/paraphrase-dataset2
[7]https://github.com/facebookresearch/asset
[8]https://huggingface.co/datasets/cestwc/paraphrase
[9]https://github.com/Yale-LILY/SummEval

10. *text-summ-g*: This dataset composed by adding the texts of the documents to the dataset *summ-g*, but with restriction that only the document texts serve as A, and only the summaries serve as B and C.
11. *text-summ-r*: Composed as *text-summ-g*, but with human-written reference summaries instead of the generated summaries.

The above datasets are diverse: besides paraphrased sentences, there are paraphrased questions (cestwc/quora), sentence simplifications (asset), related paragraphs - as summaries created from the same text (summ-g and summ-r), and related text-summary pairs (text-summ-g and text-summ-r). In order to have a manageable numbers of triplets, for some known datasets we took subsets or only parts of them - see above *catasaurus* and *cestwc*.

For comparison, we also evaluate SGPT sentence embeddings[10] Muennighoff (2022) in the same way as the neural embeddings. To our knowledge SGPT is at or near the state of the art embeddings for semantic search, hence it should conform well to the (ABC) condition Eq.2.

Table 1 contains the results of this evaluation.

dataset	embedding	error	total	wrong	I	same	diff
mrpc	SGPT	7.9e-5	24472808	1945		0.86	0.18
	Neural	6.1e-5		1491	0.78	0.54	0.22
sts	SGPT	1.0e-3	455624	460		0.85	0.09
	Neural	1.0e-3		455	0.69	0.53	0.03
catasaurus	SGPT	6.3e-5	99980000	6308		0.97	0.19
	Neural	7.0e-5		6998	0.88	0.79	0.02
asset/t	SGPT	9.7e-5	155511620	15028		0.86	0.13
	Neural	4.8e-4		74731	0.31	0.51	0.01
cestwc/t	SGPT	1.9e-2	224550000	4282171		0.68	0.15
	Neural	4.2e-2		9509234	0.42	0.31	0.03
cestwc/q	SGPT	2.1e-3	224550000	479294		0.77	0.22
	Neural	1.0e-2		2280159	0.43	0.38	0.07
cestwc/c	SGPT	4.4e-2	224550000	9920527		0.60	0.17
	Neural	1.1e-1		24563942	0.57	0.23	0.08
text-summ-g	SGPT	8.0e-3	2861100	22771		0.70	0.22
	Neural	1.5e-5		44	0.45	0.42	0.03
text-summ-r	SGPT	1.1e-2	1197900	12967		0.71	0.21
	Neural	6.7e-3		7988	0.16	0.27	0.03
summ-g	SGPT	6.3e-3	45777600	287978		0.78	0.25
	Neural	4.4e-3		203275	0.20	0.45	0.03
summ-r	SGPT	1.6e-2	11979000	190809		0.66	0.18
	Neural	3.7e-2		440827	0.34	0.22	0.02

Table 1: Performance of embeddings, measured as the ability to conform to the ABC-relation by Eq.2. Column *embedding* is the kind of embeddings: $SGPT$ or $Neural$. Column *error* is the fraction of broken ABC-relations. Column $total$ is the number of the triplets, and column $wrong$ is the number of broken triplets; $error = wrong/total$. I is the intersection of $SGTP$ and $Neural$ errors, by Eq.3. Column *same* is the average value of embeddings product for semantically similar AB texts; column *diff* is the product for semantically different AC. A row is marked green when $Neural$ error is lower than $SGPT$ error. The result for 'text-summ-g' is sensitive to random seed and the count $wrong = 44$ can shift up to 10 units up or down.

The *embedding* column shows the kind of embeddings used. Neural embeddings are obtained using the BERT-base-uncased model, from the layers listed in Section 2 and micro-tuning with the blueprints $[(2,1),(1,1),(1,2),(1,3)]$, 10 epochs and learning rate 0.01, as described in Section 2.

The *error* column shows the fraction of 'broken' triplets (A,B,C) - the triplets for which Eq.2 is not satisfied. The *intersect* column I shows the fraction of the broken triplets that the tested embeddings

[10]https://github.com/Muennighoff/sgpt

have in common with SGPT:

$$I = \frac{T_{sgpt} \bigcap T_{neural}}{min(|T_{sgpt}|, |T_{neural}|)} \tag{3}$$

Here I is the intersect, T_{sgpt} is a set of the triplets broken by SGPT embeddings, and T_{neural} is a set of the triplets broken by the neural embeddings. The intersect is not listed for SGPT itself, since it equals 1.

The *same* column shows the average value of the scalar product $(E_A * E_B)$ over all the triplets, i.e. the average value of the product between embeddings of the texts which were selected as semantically close (same group). Similarly, the *diff* column shows the average value of the scalar product $(E_A * E_C)$ over all the triplets, i.e. the average value of the product between embeddings of the texts which were selected as semantically different (different groups).

In most datasets SGPT and neural embeddings behave differently. Their errors are largely disjoint, with the overlap I below 50% for the majority of the datasets in Table 1. Another difference between SGTP and neural embeddings is how they spread out. The average product of embeddings is higher for SGPT embeddings than for neural embeddings. This is true both for semantically similar and for semantically different texts - see the *same* and *diff* columns.

3.2 Agreement with positively and negatively annotated pairs

The datasets containing the pairs of sentences deemed not similar provide another way of evaluation: any similar pair must have higher product of embeddings than any non-similar pair:

$$sim(A, B) > sim(C, D) \iff (E_A * E_B) > (E_C * E_D) \tag{4}$$

Table 2 contains the results of this evaluation.

dataset	embedding	error	total	wrong	I	same	diff
mrpc	SGPT	2.4e-1	2953956	695606		0.86	0.76
	Neural	2.6e-1		772375	0.62	0.54	0.42
sts	SGPT	4.1e-2	180492	7332		0.85	0.48
	Neural	9.9e-2		17893	0.56	0.53	0.29

Table 2: Performance of embeddings, measured as the ability to grade a pair of sentences labeled similar higher than a pair of sentences labeled as non-similar. The pair is graded by dot product of the embeddings of the sentences in the pair. Column *embedding* is the kind of embeddings: $SGPT$ or $Neural$. Column *error* is the fraction of broken cases where a pair of similar sentences was graded lower than a pair of non-similar sentences. Column $total$ is the total number of tuples (similar pair - non-similar pair). Column $wrong$ is the number of errors, so that $error = wrong/total$. I is the intersection of $SGTP$ and $Neural$ errors, by Eq.3. Column *same* is the average value of embeddings product for pairs labeled as similar; column *diff* is the average product for pairs labeled as different.

MRPC 'train' dataset contains 2474 similar and 1194 non-similar pairs. The STS 'test' dataset contains 374 pairs with score 4 or 5 - which we used as similar, and 479 pairs with score 2 or 1 - which we used as non-similar.

3.3 Correlations with human scores

If dataset contains human-scored pairs of texts or sentences, with the score reflecting at least in some sense a similarity between the texts, then the product of embeddings should correlate with the score. In Table 3 we show examples of such correlations.

The 'sts' (STS test dataset) is scored by similarity of the phrases. The 'summeval' (SummEval dataset Fabbri et al. (2021)) has human-scored, by 3 expert, each of 1700 summaries, which were generated by 17 models for 100 documents. Each summary is scored by 3 experts for four qualities - coherence, consistency, fluency and relevance. The higher the score, the closer (in sense of the quality) the summary to the text. We are using the score averaged over the 3 expert scores.

dataset	score	embedding	kendall-c	kendall-b	spearman
sts	similarity	SGPT	0.608	0.612	0.795
		Neural	0.478	0.482	0.658
summeval	coherence	SGPT	0.156	0.151	0.216
		Neural	0.100	0.097	0.137
summeval	consistency	SGPT	0.011	0.017	0.022
		Neural	0.098	0.157	0.200
summeval	fluency	SGPT	-0.004	-0.006	-0.007
		Neural	0.063	0.084	0.109
summeval	relevance	SGPT	0.198	0.196	0.275
		Neural	0.190	0.188	0.263

Table 3: Correlations of embedding product with human scores. The rows with stronger neural correlations are marked green.

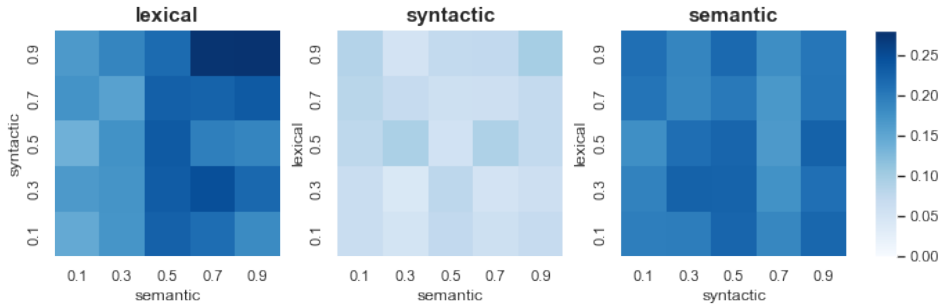

Figure 5: Evaluation of neural embeddings on generated phrases with controlled similarity. The original phrases are from MRPC. Kendall Tau (variant c) correlations of neural embeddings product with generated phrase similarity - lexical similarity on the left heatmap, syntactic similarity on the middle heatmap, and semantic similarity on the right heatmap. The X and Y axes show the similarity by the fixed qualities. For example, on the left heatmap we observe that for the phrases generated at semantic similarity parameter 0.7 and at syntactic similarity 0.9, the correlation of neural embedding product with lexical similarity is especially high.

3.4 Correlations with semantic, lexical and syntactic similarities

Using the work Bandel et al. (2022) and their provided quality-controlled paraphrase generation[11], we can review how neural embeddings correlate with semantic, lexical and syntactic similarities of paraphrases. We generate paraphrases with different similarities by one of these three quality-control parameters, while keeping two other qualities fixed.

For the evaluation, we have 100 original phrases; for each of the original phrase we generate 20 phrases, corresponding to changing the control parameter for selected quality in the range $0.05, 0.10, 0.15, ..., 0.95$. This could give 2000 generated phrases, each with its own 'score' (in the range $0.05 - 0.95$). However, as it happens occasionally, the increase of the control parameter by 0.05 sometimes generates the same phrase, - which we ignore. Therefore in the final results there are around 300 - 900 (rather than 2000) scored phrases in each evaluation, depending on the selected quality and on the values of the fixed qualities (see the numbers in Appendix B). Our own score of a generated phrase is simply a scalar product of the neural embeddings of the generated phrase and of the original phrase.

As our original phrases we take first sentences from the first 100 pairs of phrases from MRPC train dataset. The correlations obtained for the gererated phrases are shown in Figure 5.

Naturally, all the correlations are positive, and are especially strong for lexical and semantic qualities. The correlations are also all positive for SGPT embeddings. Since all the correlations are positive, we can visualize and compare the goodness of neural embeddings vs SGPT by the ratio of neural

[11]https://github.com/IBM/quality-controlled-paraphrase-generation

correlations to the SGPT correlations. These ratios are shown in Figure 6. Accordingly to Figure

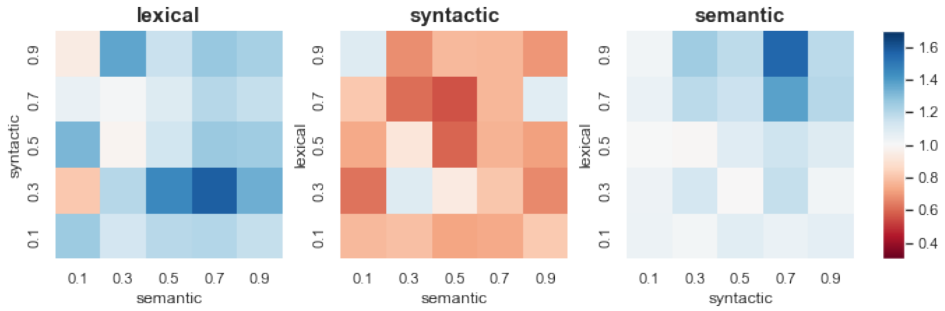

Figure 6: Ratio of Kendall Tau correlations of neural embeddings to SGPT embeddings. (For phrases control-generated from MRPC.)

6, neural embeddings products correlate better (than SGPT embeddings products) with semantic similarity, and, in most cases, with lexical similarity. SGPT is better, in most cases, for syntactic similarity. Stronger correlation with semantic similarity is unambiguously good for an embedding. Lexical and syntactic similarities are supposedly (at least approximately) untangled from semantics, so a stronger correlation with them may be considered as a desired or not desired feature, depending on downstream goals.

MRPC sentences are typically long. We repeat our evaluation on short sentences, by selecting from STS test subset 100 first sentences of length between 20 and 40 characters. We generated similar phrases in the same manner as for the MRPC phrases. The resulting correlations of neural embedding products with similarities of the generated phrases are shown in Figure 7.

The corresponding ratio of neural embeddings correlations to SGPT correlations is shown in Figure 8. Again, neural embeddings have stronger correlations with semantic and lexical similarities.

3.5 Discussion

As we observed, by some criteria and datasets, neural embeddings are better, and by other criteria and datasets SGPT embeddings are better. Here we briefly discuss the observations and make several conclusions and speculations.

The average products of neural embeddings are consistently much lower than the products of SGPT embeddings: lower for similar phrases (or texts), and lower for non-similar phrases (or texts) - see Tables 1, 2. This may be related to the fact that in most our evaluations the intersection of neural and

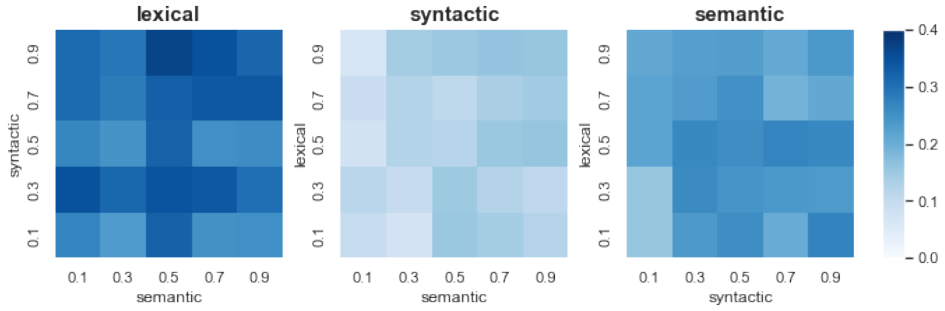

Figure 7: Evaluation of neural embeddings on short generated phrases with controlled similarity. The original phrases are from STS. Kendall Tau (variant c) correlations of neural embeddings product with generated phrase similarity - lexical similarity on the left heatmap, syntactic similarity on the middle heatmap, and semantic similarity on the right heatmap. The X and Y axes show the similarity by the fixed qualities.

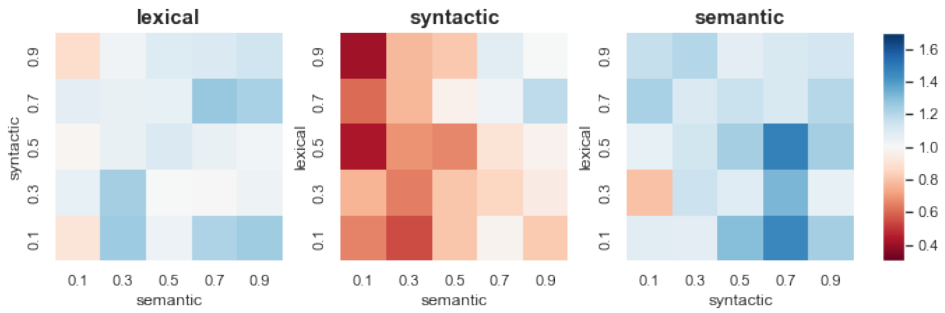

Figure 8: Evaluation on short sentences. Ratio of Kendall Tau correlations of neural embeddings to SGPT embeddings. (For phrases control-generated from STS.)

SGPT errors is low. Interestingly, MRPC and STS datasets were easier for neural embeddings by the triplets criteria (Table 1), but got easier for SGPT for the positive-negative pairs criteria (Table 2). As an illustration of the difference between neural and SGPT embeddings, in Appendix C we show results for simplest 'ensemble': the concatenation of the two. As expected, in many cases it outperforms them both.

Neural embeddings happen to better represent texts when two compared texts are different by style - as for text-summary pairs (with either generated or refernce summaries) and for groups of generated summaries (Table 1). But SGPT got better for groups of reference summaries. Since the reference summaries may have more difference in their composition structure, we speculate that this relates to two other observations: First, in Table 3, the neural embeddings are weaker by coherence correlations, while being almost the same by relevance and better by consistency and fluency correlations. Second, neural embeddings correlate weaker than SGPT with syntactic quality of generated phrases (Figures 6 and 8), while being stronger with semantic and lexical qualities.

4 Conclusion

In this paper we introduced *neural embeddings*, a new method for representing text as vectors. We observed that while the performance of these embeddings is comparable to that of SGPT embeddings, about half of the errors are different - depending on the dataset. Based on our observations in Section 3.4 we speculate that neural embeddings make more emphasis on semantic and lexical qualities, and less on syntactic qualities.

To our knowledge, this is the first attempt to obtain representations in this manner; in principle the technique may not be restricted to text.

References

Samuel Yang Alexander Liu. Masked autoencoders as the unified learners for pre-trained sentence representation. *arXiv*, arXiv:2208.00231, 2022. URL https://arxiv.org/abs/2208.00231.

Fernando Alva-Manchego, Louis Martin, Antoine Bordes, Carolina Scarton, Benoît Sagot, and Lucia Specia. ASSET: A dataset for tuning and evaluation of sentence simplification models with multiple rewriting transformations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4668–4679, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.424. URL https://aclanthology.org/2020.acl-main.424.

Elron Bandel, Ranit Aharonov, Michal Shmueli-Scheuer, Ilya Shnayderman, Noam Slonim, and Liat Ein-Dor. Quality controlled paraphrase generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 596–609, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.45. URL https://aclanthology.org/2022.acl-long.45.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2001. URL https://aclanthology.org/S17-2001.

Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljacic, Shang-Wen Li, Scott Yih, Yoon Kim, and James Glass. DiffCSE: Difference-based contrastive learning for sentence embeddings. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4207–4218, Seattle, United States, July 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.naacl-main.311.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. URL https://aclanthology.org/I05-5002.

Alexander R. Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. SummEval: Re-evaluating summarization evaluation. *Transactions of the Association for Computational Linguistics*, 9:391–409, 2021. URL https://doi.org/10.1162/tacl_a_00373.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6894–6910, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.552. URL https://aclanthology.org/2021.emnlp-main.552.

John Giorgi, Osvald Nitski, Bo Wang, and Gary Bader. DeCLUTR: Deep contrastive learning for unsupervised textual representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 879–895, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.72. URL https://aclanthology.org/2021.acl-long.72.

Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In Aasa Feragen, Marcello Pelillo, and Marco Loog (eds.), *Similarity-Based Pattern Recognition*, pp. 84–92, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24261-3.

James Y. Huang, Kuan-Hao Huang, and Kai-Wei Chang. Disentangling semantics and syntax in sentence embeddings with pre-trained language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1372–1379, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.108. URL https://aclanthology.org/2021.naacl-main.108.

Philip May. Machine translated multilingual sts benchmark dataset. 2021. URL https://github.com/PhilipMay/stsb-multi-mt.

Niklas Muennighoff. Sgpt: Gpt sentence embeddings for semantic search. *arXiv*, arXiv:2202.08904, 2022. URL https://arxiv.org/abs/2202.08904.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language*

Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410.

Oleg Vasilyev, Vedant Dharnidharka, and John Bohannon. Fill in the BLANC: Human-free quality estimation of document summaries. In Steffen Eger, Yang Gao, Maxime Peyrard, Wei Zhao, and Eduard Hovy (eds.), *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*, pp. 11–20. Association for Computational Linguistics, November 2020. doi: 10.18653/v1/2020.eval4nlp-1.2. URL https://aclanthology.org/2020.eval4nlp-1.2.

Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1386–1393, 2014.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics (2020), 2020. URL https://aclanthology.org/2020.emnlp-demos.6.

Yuanmeng Yan, Rumei Li, Sirui Wang, Fuzheng Zhang, Wei Wu, and Weiran Xu. ConSERT: A contrastive framework for self-supervised sentence representation transfer. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5065–5075, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.393. URL https://aclanthology.org/2021.acl-long.393.

Dejiao Zhang, Shang-Wen Li, Wei Xiao, Henghui Zhu, Ramesh Nallapati, Andrew O. Arnold, and Bing Xiang. Pairwise supervised contrastive learning of sentence representations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5786–5798, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.467. URL https://aclanthology.org/2021.emnlp-main.467.

Yan Zhang, Ruidan He, Zuozhu Liu, Kwan Hui Lim, and Lidong Bing. An unsupervised sentence embedding method by mutual information maximization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1601–1610, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.124. URL https://aclanthology.org/2020.emnlp-main.124.

A FASTER MICRO-TUNING

For convenience or reproducibility, the neural embeddings can be obtained simply by using the BERT model as it is, with all the layers frozen, except the selected layers, as described in Section 2. But there are several simplifications that make obtaining neural embeddings almost as fast as a simple forward run of BERT.

Since the micro-tuning 'dataset' is extremely small, all the inputs can be kept on GPU through all epochs. More importantly, since all the selected (for embeddings) layers are located at the top of the model, there is no need to repeat the forward run through all the lower transformer blocks. The hidden states from the last block preceding the layers of interest can be obtained only once, at the first epoch, and then reused in all the next epochs. In result, at each epoch except the first one, the micro-tuning deals only with the top of the model (separated from the rest). Again, since the micro-tuning 'dataset' is extremely small, even these hidden states can be kept on GPU, unless we are dealing with a really long text of too many input windows.

For example, using simply the whole BERT model (with 10 epochs), the average time of obtaining neural embedding on GPU P100 for MRPC sentence is 0.24 sec., and for STS sentence is 0.16 sec.

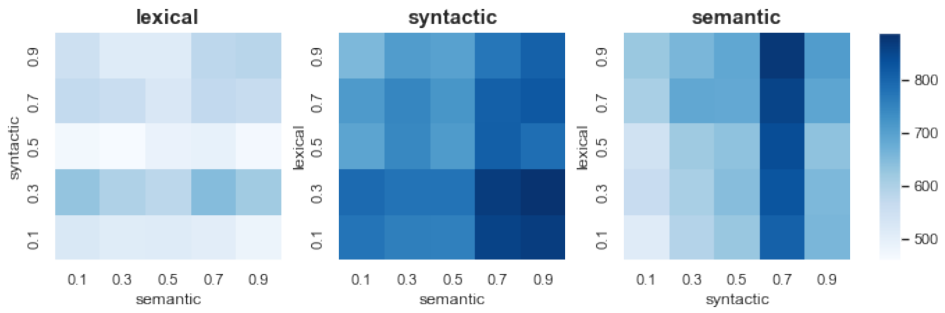

Figure 9: Number of control-generated phrases - for lexical similarity (left), syntactic similarity (middle), and semantic similarity (right). The X and Y axes show the similarity by the fixed qualities. The original phrases are from MRPC.

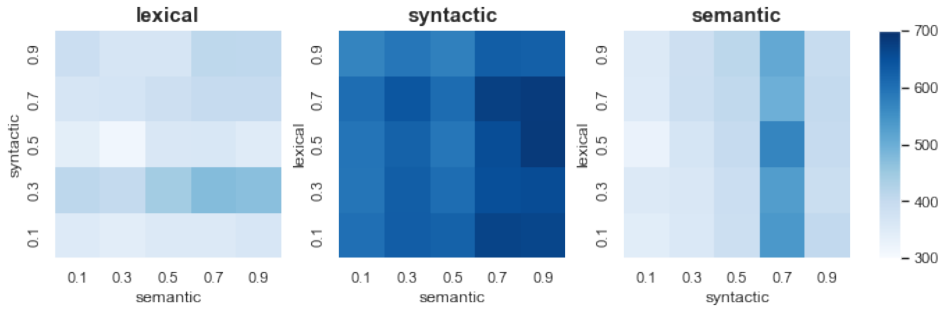

Figure 10: Number of control-generated phrases - for lexical similarity (left), syntactic similarity (middle), and semantic similarity (right). The X and Y axes show the similarity by the fixed qualities. The original phrases are from STS.

With hidden states reused, these times become 0.096 sec. and 0.057 sec. Reducing the number of epochs offers another option: Switching from 10 to 5 epochs decreases the time almost twice, with results in most cases worse only but several percents.

Having the selected layers on the very top gives the advantage of processing everything below only at the first epoch. I principle, we can use the following layers of the last transformer block:

1. $L_1 = bert.encoder.layer.11.output.dense.bias$
2. $L_2 = bert.encoder.layer.11.output.LayerNorm.weight$
3. $L_3 = bert.encoder.layer.11.output.LayerNorm.bias$

These are the same layers that we used through the paper, but taken from the last transformer block (#11) rather than from the top ('cls') of the BERT model. Using these layers gives similar results, but makes the processing slower.

B CONTROLLED-GENERATED PHRASES

In section 3.4 we used control-generated phrases for comparing lexical, syntactic and semantic representation of the phrases by neural embeddings. Since increase of a control parameter by small increment sometimes caused generation of the same phrase, we skipped the duplicates. Here we show the number of the generated phrases, in Figures 9 and 10. At any choice of the fixed control parameters, the number of generated phrases not less than 300. All correlations shown in Section 3.4 have p-value far below 0.05.

C NEURAL AND SGPT EMBEDDINGS CONCATENATED

We observed in Section 3 that neural embeddings and SGPT behave differently: their errors do not overlap greatly, their average products have different ranges, and their correlations with text qualities are different. Here we suggest an additional illustration of this point. We create a simple 'ensemble' by concatenating these embeddings into a single embedding. An ensemble is expected to perform better than either of its components if the they perform not too differently but have different properties.

In Table 4 we observe that this is indeed true. The exception is 'text-summ-g' where neural embedding was orders of magnitude better, 'cestwc/c' with almost a tie, and, strangely, catasaurus. In Table 5 for 'sts', SGPT was much better to start with. In Table 3 the neural and SGPT embeddings were almost equally good correlating with relevance, and in that case the concatenation is much stronger than either of them.

dataset	embedding	error	total	wrong	I	same	diff
mrpc	SGPT	7.9e-5	24472808	1945		0.86	0.18
	Neural	6.1e-5		1491	0.78	0.54	0.22
	Neural+SGPT	5.7e-5		1394	0.90	0.70	0.10
sts	SGPT	1.0e-3	455624	460		0.85	0.09
	Neural	1.0e-3		455	0.69	0.53	0.03
	Neural+SGPT	9.0e-4		412	0.86	0.69	0.06
catasaurus	SGPT	6.3e-5	99980000	6308		0.97	0.19
	Neural	7.0e-5		6998	0.88	0.79	0.02
	Neural+SGPT	6.8e-5		6751	0.91	0.88	0.10
asset/t	SGPT	9.7e-5	155511620	15028		0.86	0.13
	Neural	4.8e-4		74731	0.31	0.51	0.01
cestwc/t	SGPT	1.9e-2	224550000	4282171		0.68	0.15
	Neural	4.2e-2		9509234	0.42	0.31	0.03
	Neural+SGPT	1.7e-2		3775736	0.80	0.50	0.09
cestwc/q	SGPT	2.1e-3	224550000	479294		0.77	0.22
	Neural	1.0e-2		2280159	0.43	0.38	0.07
	Neural+SGPT	2.0e-3		439138	0.66	0.57	0.15
cestwc/c	SGPT	4.4e-2	224550000	9920527		0.60	0.17
	Neural	1.1e-1		24563942	0.57	0.23	0.08
	Neural+SGPT	4.4e-2		9985684	0.82	0.42	0.13
text-summ-g	SGPT	8.0e-3	2861100	22771		0.70	0.22
	Neural	1.5e-5		44	0.45	0.42	0.03
	Neural+SGPT	3.7e-4		1068	1.00	0.56	0.12
text-summ-r	SGPT	1.1e-2	1197900	12967		0.71	0.21
	Neural	6.7e-3		7988	0.16	0.27	0.03
	Neural+SGPT	4.4e-3		5238	0.94	0.49	0.12
summ-g	SGPT	6.3e-3	45777600	287978		0.78	0.25
	Neural	4.4e-3		203275	0.20	0.45	0.03
	Neural+SGPT	2.6e-3		118566	0.93	0.61	0.14
summ-r	SGPT	1.6e-2	11979000	190809		0.66	0.18
	Neural	3.7e-2		440827	0.34	0.22	0.02
	Neural+SGPT	1.2e-2		143632	0.86	0.44	0.10

Table 4: Performance of embeddings, measured as the ability to conform to the ABC-relation by Eq.2. This duplicates Table 1, with addition of concatenated embedding Neural+SGPT. A row with lowest error is marked green.

Interesting, even though evaluation of summary quality was not the purpose of the embeddings, neural embeddings correlations for consistency scores exceed correlations of all flavors of ROUGE (ROUGE-L, ROUGE-1, ROUGE-2 and ROUGE-3). Similarly, Neural+SGPT correlations are higher than all flavors of ROUGE for relevance, and SGPT correlations are higher for coherence.

dataset	embedding	error	total	wrong	I	same	diff
mrpc	SGPT	2.4e-1	2953956	695606		0.86	0.76
	Neural	2.6e-1		772375	0.62	0.54	0.42
	Neural+SGPT	2.3e-1		685446	0.77	0.70	0.59
sts	SGPT	4.1e-2	180492	7332		0.85	0.48
	Neural	9.9e-2		17893	0.56	0.53	0.29
	Neural+SGPT	4.8e-2		8735	0.70	0.69	0.38

Table 5: Performance of embeddings, measured as the ability to grade a pair of sentences labeled similar higher than a pair of sentences labeled as non-similar. This duplicates Table 2, with addition of concatenated embedding Neural+SGPT. A row with lowest error is marked green.

dataset	score	embedding	kendall-c	kendall-b	spearman
sts	similarity	SGPT	0.608	0.612	0.795
		Neural	0.478	0.482	0.658
		Neural+SGPT	0.592	0.596	0.779
summeval	coherence	SGPT	0.156	0.151	0.216
		Neural	0.100	0.097	0.137
		Neural+SGPT	0.156	0.151	0.215
summeval	consistency	SGPT	0.011	0.017	0.022
		Neural	0.098	0.157	0.200
		Neural+SGPT	0.068	0.108	0.138
summeval	fluency	SGPT	-0.004	-0.006	-0.007
		Neural	0.063	0.084	0.109
		Neural+SGPT	0.038	0.051	0.065
summeval	relevance	SGPT	0.198	0.196	0.275
		Neural	0.190	0.188	0.263
		Neural+SGPT	0.233	0.231	0.324

Table 6: Correlations of embedding product with human scores. This duplicates Table 2, with addition of concatenated embedding Neural+SGPT. A row with highest correlations is marked green.