Sarcasm Retrieval in fine-tuned Sentence Embeddings

Jens Breitung
Computer Science / 2023
jensfb@stud.ntnu.no

Abstract

In recent years, sarcasm detection has received a lot of attention in the context of sentiment analysis, with much less research focusing on sarcasm generation. In this paper, we explore the possibility of sarcasm retrieval from large collections of unlabelled sentences. Specifically, we investigate the encoding of sarcastic polarity into sentence embeddings in order to increase the likelihood of retrieving sentences with the same polarity as the input sentence while maintaining semantic similarity. Our experiments involve fine-tuning the pre-trained sentence embedding 'all-miniLM-L6-v2' using conventional loss functions used to train sentence embeddings. To measure the performance of the resulting models we introduce two new metrics: Polarity Score and Semantic Similarity Score. Our findings show that perfect achievement of both polarity and semantic similarity is not possible due to them being opposing objectives. However, accepting a slight decrease in semantic similarity significantly improves polarity of retrieved sentences, indicating the overall viability of this novel approach.

1 Introduction

Sarcasm is a type of speech where the intended meaning is opposite to what is expressed, often used to convey criticism or humor indirectly. Context and non-verbal cues play a crucial role in detecting sarcasm in human interactions, and in social media often it is often denoted by indicators such as '#sarcasm' on Twitter or '\s' on Reddit. While research on sarcasm detection in sentiment analysis has been extensive (Joshi et al. (2016)), sarcasm generation has received much less attention with existing models being very complex in their architecture (cf. Mishra et al. (2019)). General purpose models such as GPT-3 exhibit capabilities of sarcasm generation, however, training requires prohibitively large amounts of data.

In this paper, we propose a middle ground between text generation and binary classification by exploring the possibility of sarcasm "retrieval" using sentence embeddings. State-of-the-art sentence embeddings capture semantic similarity of sentences but do not inherently encode sarcastic polarity. We aim to incorporate information about sarcasm into sentence embeddings to increase the likelihood of retrieving sentences with the same polarity as the input sentence, all while maintaining semantic similarity. This conceptually allows for the design of a "sarcasm retrieval system" that can suggest sarcastic texts similar to a single given sarcastic input sentence by leveraging large amounts of unlabelled data.

The remainder of this paper is structured as follows: In Section 2 we introduce the theoretic background for our work, followed by a brief overview over related work in Section 3. The architecture of our model is presented in Section 4, the experimental setup as well as our results are found in Section 5. A more in-depth discussion follows in Section 6. A conclusion as well as avenues for future work are given in Section 7.

2 Background

In the following, we will give a brief introduction to the terminology used in this paper as well as give an overview of the theoretical foundations that our approach underlies.

2.1 Sentence Embeddings and Cosine Similarity

To allow machine learning algorithms to operate on texts, they are split into words or more generally tokens. These can then be represented as one-hot encoded vectors where the single non-zero position is the tokens index in the vocabulary. This representation is extremely lossy because all information of semantic meaning is lost. Word embeddings are a way to represent words in a much lower dimensional space such that words which are semantically similar are close together while words that are dissimilar are far apart. Here, closeness between two words is measured using cosine similarity, defined as

$$\text{cos\_sim}(w_1, w_2) := \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \cdot \|\mathbf{x}_2\|},$$

where $\mathbf{x}_i$ is the vector embedding of the word w_i. Sentence embeddings have the same underlying concept, however the unit on which they operate are sequences of words instead of individual ones. One way to do this, is to compute weighted attention scores of the individual words (by using transformers) followed by aggregation ('pooling'), e.g. taking the arithmetic mean. In the following, for sentences s_1, s_2 we denote by $\text{cos\_sim}_E(s_1, s_2)$ their cosine similarity with respect to the sentence embedding E.

2.2 Loss Functions for Sentence Embeddings

To improve the semantic quality of sentence embeddings they are trained with different loss functions depending on what properties are needed for downstream tasks. The loss functions relevant to our work are the following:

- **TripletLoss** (Schroff et al. (2015)): To use this loss function, training data consists of triples of sentences (A, P, N) where A is the 'anchor' and P is a positive sentence (similar to the anchor) and N is a negative one. Then the loss is expressed as $\max(\|\text{emb}(A) - \text{emb}(P)\| - \|\text{emb}(A) - \text{emb}(N)\| + \lambda, 0)$ where λ is the margin, specifying the minimum separation between A and N how far N needs to be separated from A at the very least.

- **(Online)ContrastiveLoss** (Hadsell et al. (2006)): Training data consists of $\{0, 1\}$-labelled tuples of sentences where the label indicates whether $\|\text{emb}(S_1) - \text{emb}(S_2)\|$ is to be minimized or maximized. In the online version, the loss is only calculated for hard positive and hard negative pairs, that is, pairs of similar sentences that are far apart and pairs of opposites that are very close. The margin parameter λ controls how far dissimilar pairs need to be separated.

- **MultipleNegativesRankingLoss** (Henderson et al. (2017)): Given a sequence of pairs of similar sentences, it assumes that sentences of distinct pairs are dissimilar. For a given first sentence from one of the pairs, it tries to maximize the probability of picking the corresponding second sentence from from all second sentences in the sequence.

3 Related Work

Transformers were first introduced in the paper 'Attention is All you Need' (Vaswani et al. (2017)). Ever since they have been used in increasingly varying applications (Lin et al. (2021)). However, its attention masking can be problematic for tasks in which context from words (or tokens) later in the sequence are important, e.g. question answering. To this end, Devlin et al. proposed bidirectional transformer networks. Their language representation model 'BERT' has since become a baseline for a multitude of NLP research projects (Min et al. (2021)), most notably 'RoBERTa' which uses the same underlying architecture but with increased training data and duration as well as a 'on the fly' masking of tokens (Liu et al. (2019)). All of these approaches operate on a word or token level and produce word embeddings similar to word2vec (Mikolov et al. (2013)) or GloVe (Pennington et al. (2014)) and are thus not well suited for tasks such as semantic textual similarity (STS) as they lack the ability to capture semantic meaning of texts. According to Reimers and Gurevych, the most common way of obtaining a sentence embedding from BERT is to average the BERT output layer ('BERT' embedding). However, they find that the resulting fixed-size sentence embeddings often perform worse than averaging of GloVe vectors.

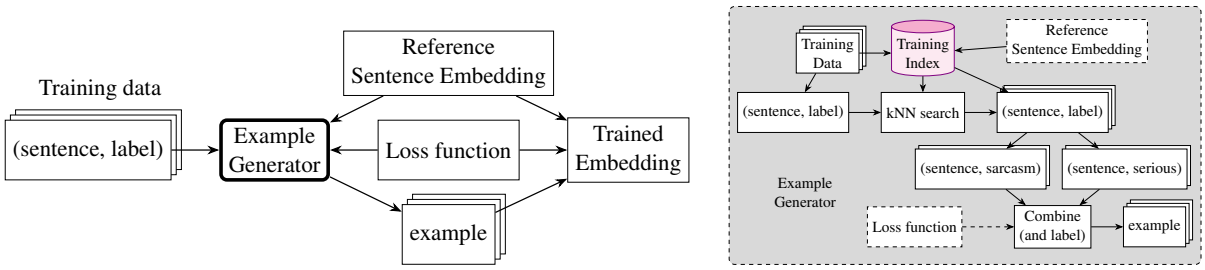
For semantic similarity comparison, BERT therefore relies on similarity scoring for pairs of texts. For large numbers of texts, computing all pairwise scores quickly becomes computationally infeasible. By modification of the BERT network, it is possible to compute sentence embeddings that capture semantic meaning (Reimers and Gurevych (2019)) and thus allows for comparison of sentences via cosine similarity. To the best of our knowledge these recent advances have not yet been studied in the context of sarcasm detection or retrieval.

4 Architecture

Having surveyed the most recent and relevant work to our project, in this section we describe our architecture in detail. Conceptually, it is built around a 'reference' sentence embedding which we use as a baseline and then fine-tune. For clarity, we split the architecture description into two stages, a training phase and an inference phase which we elaborate on in the following.

4.1 Training phase

The training phase is illustrated in Figure 1 and summarised in Figure 1a. Given labelled sentences (sarcastic or not), we generate a new set of examples using the Example Generator (see Figure 1b). These examples are then used to fine-tune the reference embedding, resulting in the trained embedding. To generate examples from a labelled sentence (s, y), we embed the entire training data using the reference embedding and query the k nearest neighbours. The resulting sentences are split by polarity and combined with s to create labelled tuples or triples, as discussed in Section 2. The shape of a valid example depends on the loss function used, leading to varying numbers of training examples. To address this, we randomly discard examples using dropouts for each loss function, which allows for more diverse examples compared to simply truncating the list of all possible examples.



(a) The overall training architecture

(b) The Example Generator in more detail

Figure 1: Training phase

4.2 Inference phase

After training, all available (possibly unlabelled) sentences are embedded using the trained embedding and saved to an index for efficient lookup. When given an incoming sentence, the inference model computes its embedding and returns the k nearest neighbors as suggestions, as shown in Figure 2.

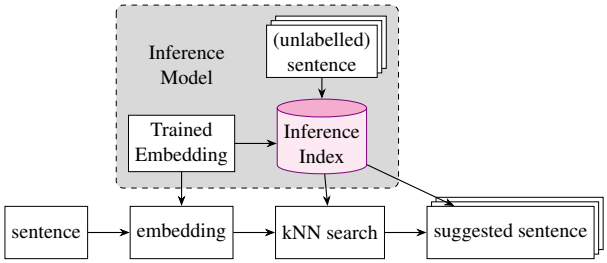


Figure 2: Inference Phase

5 Experiments and Results

As a baseline for our experiments, we use the existing sentence embedding "all-MiniLM-L6-v2" obtained from Hugging Face.[1] Table 1 displays the loss functions used for fine-tuning this embedding, with the exact implementations taken from the Python library 'sentence-transformers'.[2] The full dataset, discussed in more detail below, is split into two equal-sized subsets, one for training and one for evaluation. This unconventional split is chosen because instead of having labelled examples with known targets, our goal is to retrieve other sentences with the same label from the search space. Therefore, the splits should be roughly equal in size to ensure enough available data in both cases. From the test data, we initially randomly sample a set T of $n = 100$ sentences, which will be used for evaluating model performance. Our experiment is conducted in iterations, with each iteration consisting of two stages:

1) **Training**: We sample $n = 5000$ sentences from the train data and generate examples by retrieving the $k = 64$ nearest neighbours of each sentence (not including the sentence itself) and combining them into valid examples for the loss functions as described in Section 4. Then the embeddings are trained on the examples with respect to the loss function for one 'epoch', i.e. each example is used exactly once.

2) **Evaluation**: We build the inference models for the embeddings by embedding the entirety of the training data and creating an index. The resulting inference models are then evaluated on T. How this is done exactly is described in the next subsection.

We perform this two-step process for a total of ten iterations, reusing the embeddings obtained from the previous iteration. However, training examples are still sampled using the unaltered initial embedding, R. Training of a single model for ten iterations took roughly 20 to 30 minutes.

Loss Function	Hyperparameters used	Dropout
Triplet Loss	$\lambda \in \{0.01, 0.05, 0.1, 0.5, 2, 5\}$	0.98
Multiple Negatives Ranking Loss	–	0.7
Contrastive Loss	$\lambda \in \{0.2, 0.5\}$	0.8
Online Contrastive Loss	$\lambda = 0.5$	0.8

Table 1: Overview of loss functions and their hyperparameters used during training. The dropout value indicates with which probability a potential example was discarded by the Example Generator.

5.1 Model evaluation

To measure the performance of an inference model M we compare it to the reference model R, that is, an inference model using the 'all-miniLM-L6-v2' as the sentence embedding. For a given sentence s we retrieve the k nearest neighbours (we chose $k = 64$) using both models ($s_1^M, \ldots, s_k^M$ and $s_1^R, \ldots, s_k^R$ respectively, sorted from closest to furthest) and evaluate them on the criteria of sarcastic polarity and semantic similarity. The precise formulation of these two scores is described below.

5.1.1 Polarity Score

To measure whether a model favours texts of the same polarity as the input in its suggestions we compute a weighted average polarity score over the k suggestions depending on the polarity of s. Formally, for a sentence s this can be expressed as

$$\mathcal{P}_M(s) := \sum_{i=1}^{k} w_i \cdot \text{pol}\big(s_i^M\big) \quad \text{where} \quad \text{pol}\big(s_i^M\big) := \begin{cases} 1 & \text{if } s \text{ and } s_i^M \text{ have the same polarity,} \\ 0 & \text{otherwise.} \end{cases}$$

[1] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[2] https://www.sbert.net

The weights w_i can be chosen to reflect the importance of ranked suggestions. Instead of averaging them, we choose a linear discounting model where the i-th suggestion is scaled by a factor of $k + 1 - i$. By normalization we get weights $w_i := \frac{2 \cdot (k+1-i)}{k \cdot (k+1)}$. If the suggestions are mostly of the same polarity as the input, this is reflected in a value close to one. In any case, we would expect $\mathcal{P}_M(s) > \mathcal{P}_R(s)$, i.e. our model is better at suggesting sentences of the same polarity than the reference model.

5.1.2 Semantic Similarity Score

Of course having suggestions of matching polarity is not sufficient for them to be any good. Hence we need a way to measure semantic similarity of the outputs to a given input. To this end we compute the weighted average cosine similarity between the suggested sentences of M when using the embeddings of R for these sentences (since R is pre-trained for semantic similarity). Formally, this can be expressed as

$$\mathcal{S}_M(s) := \sum_{i=1}^{k} w_i \cdot \text{cos\_sim}_R\big(s, s_i^M\big).$$

If M suggests sentences that remain semantically similar to the input we should see that $\mathcal{S}_M(s)$ is about equal or only slightly lower than $\mathcal{S}_R(s)$, that is the semantic similarity score obtained from the samples retrieved by R itself.

Hence a 'good' model will return a large fraction of sentences with the same polarity as the input sentence while keeping the semantic similarity close to that obtained from the original embeddings.

5.2 Dataset

The dataset used for our experiments is the 'News Headlines Dataset For Sarcasm Detection' dataset[3], which contains 28,619 news headlines from 'HuffPost' and 'The Onion', a satire news website. As Misra and Arora pointed out, taking headlines from these sources guarantees high quality labels. Furthermore, headlines are mostly self-contained and do not rely on too much additional context as is the case with Tweets for example, which are often in reply to other tweets that are relevant context.

To further increase the amount of data, we extended the dataset by including headlines from 'The Babylon Bee' (satire) and 'Politico' (serious), which were scraped from their websites using the Python libraries 'Selenium' and 'BeautifulSoup'. After removing duplicates and filtering out headlines with less than 5 or more than 15 words, as well as headlines containing the substring 'Politico Playbook'[4], we ended up with a total of 53,925 labeled headlines.

Since the reference sentence embedding already handles preprocessing in its pipeline, we applied no further preprocessing to our data, except for converting non-ASCII characters to their ASCII equivalent and lowercasing the additionally scraped data to match the format of the existing dataset we expanded on.

5.3 Results

Figure 3 shows the polarity and semantic similarity scores of the inference models that is achieved by training the initial embedding using the loss functions and respective hyperparameters as shown in Table 1. The final scores of all models are also shown in Table 2. For better visual clarity we additionally provide Figures 4 and 5 where only (non-)triplet-based loss functions are shown respectively, as well as Tables 3 and 4 for a full list of measured scores.

All models exhibit an improvement in polarity but a deterioration in semantic similarity, which is expected as favouring sentences of a specific polarity increases the average distance in the reference embedding. Among the models tested, MultipleNegativesRankingLoss stands out as an outlier, achieving only a polarity score of 68.5% (a 2% improvement) compared to the 88%+ scores achieved by all other models. Furthermore, MultipleNegativesRankingLoss remains semantically similar to the reference embedding, while other models show significant declines with considerable variation.

[3] https://www.kaggle.com/datasets/rmisra/news-headlines-dataset-for-sarcasm-detection

[4] These headlines are of a specific news article series and all start with this substring. This results in all of them being relatively close to each other in the reference embedding even though they may cover very different topics.

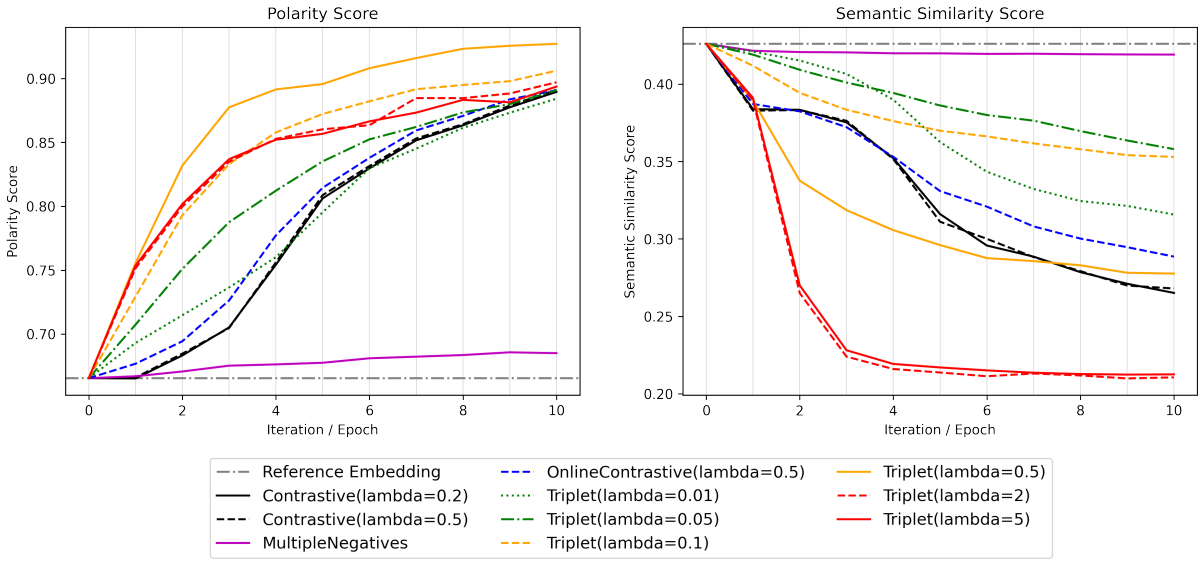


Figure 3: Training results for each model configuration when evaluated on T. The reference embedding is the unaltered 'all-MiniLM-L6-v2'.

Model Name	$\mathcal{P}_M(T)$	$\mathcal{S}_M(T)$
Reference Embedding	0.6655	0.4260
Contrastive(lambda=0.2)	0.8895	0.2651
Contrastive(lambda=0.5)	0.8912	0.2680
MultipleNegatives	0.6850	0.4191
OnlineContrastive(lambda=0.5)	0.8908	0.2886
Triplet(lambda=0.01)	0.8842	0.3157
Triplet(lambda=0.05)	0.8908	0.3580
Triplet(lambda=0.1)	0.9061	0.3530
Triplet(lambda=0.5)	0.9271	0.2776
Triplet(lambda=2)	0.8970	0.2106
Triplet(lambda=5)	0.8937	0.2125

Table 2: Polarity and semantic similarity scores of all models after ten iterations of training. The scores are obtained by averaging the individual scores for all $s \in T$.

In terms of margin values, TripletLoss and ContrastiveLoss yield similar results for the two highest margins tested. The online variant outperforms ContrastiveLoss in both metrics, regardless of the margin value chosen in the latter case (either $\lambda = 0.5$ or $\lambda = 0.2$).

Intuitively, one would expect larger margin values to result in better polarity scores. However, our experiments reveal that smaller margins lead to significantly better results in both metrics in the case of TripletLoss. Reducing λ to 0.5 (from previous values of $\lambda = 2$ or $\lambda = 5$) shows a significant improvement in both metrics. Further reduction to 0.1 only leads to a slight decrease in polarity (still higher than the largest λ-values) but another substantial improvement in semantic similarity, exceeding a value of 0.35. Lowering the margin even more to $\lambda = 0.05$ continues this trend, although the magnitude of change is smaller. Finally, $\lambda = 0.01$ does not yield any further improvement but instead performs worse in polarity and much worse than $\lambda = 0.05$ and $\lambda = 0.1$ in semantic similarity, although it still outperforms OnlineContrastiveLoss in this aspect.

All TripletLoss models (except for $\lambda = 0.01$) improve quickly in polarity early on and then flatten out after four iterations. Meanwhile, (Online)ContrastiveLoss models see almost no improvement for at least two iterations. Nevertheless, they suffer from the same initial drop in semantic similarity after just one iteration.

6 Evaluation and Discussion

With the results of our experiments presented at the end of the last section, we now discuss them in more detail. Specifically, we focus on possible reasons for the exhibited model performances as well as question our findings with respect to the quality of our data and the metrics we used.

As previously mentioned, MultipleNegativesRankingLoss stands out as an outlier in both metrics. This discrepancy is likely attributed to poor example generation for this particular loss function. The loss function treats sentences from distinct sentence pairs as dissimilar, but in our case, we generate multiple similar pairs with the *same* first sentence, resulting in contradictory examples. Since this problem does not arise for any of the other loss functions, this is the most plausible explanation for its significanlty different performance compared to all other models we tested. Our key takeaway here is that the implicit relations between distinct training examples severely restrict our flexibility in example generation. Hence, MultipleNegativesRankingLoss may be unsuitable for fine-tuning for sarcastic polarity as we do not have as much control over the targeted separations between specific sentences.

In contrast, all other loss functions have independent examples and offer a margin parameter that controls how far similar sentences of opposing polarities should be separated. Interestingly, a high margin value does not necessarily coincide with good model performance. While polarity tends to be better for larger values of λ, high margins severely deteriorate semantic similarity. This is to be expected, since a high margin 'pushes apart' sentences of opposite polarity. They may therefore end up close to sentences with entirely different semantic meanings. After a single iteration, all high margin models had a measurable drop in semantic similarity but in the case of TripletLoss also achieved by far the best polarity scores. However, our results indicate that high margins are particularly problematic when training the embeddings for multiple iterations. Over the course of as few as three iterations, high margin models lost almost all semantic meaning that was embedded in the initial sentence embedding. The main problem here clearly is the lack of balance between the two present objectives. On the one hand we want to separate sentences based on polarity while on the other remain as close to the semantic similarity of the reference model as possible. High margins seem to favour the former very strongly, hence after only few iterations we reach a severe imbalance of the two metrics.

It is however surprising that a medium value ($\lambda = 0.5$) is able to outperform the polarity of the high margin models while still being significantly better in terms of similarity. Lowering the margin hence seems to trade polarity in favour of better semantic similarity, however this relation does not hold forever with $\lambda = 0.01$ being worse than $\lambda = 0.05$ and $\lambda = 0.1$ in both metrics after half the iterations.

6.1 Data quality

Caution should be exercised when interpreting our results for less structured texts, such as Twitter data, as our dataset only contained news headlines, which are quite similar in their structure. Moreover, the dataset we used only included news headlines from four news outlets, and certain topics may be biased towards a certain polarity. For example, The Onion frequently has headlines such as 'Local man mows the lawn.' that poke fun at sensationalism in news. However, there are few semantically similar news headlines from serious news outlets. Additionally, with 42.5% of the dataset being sarcastic, the proportion of sarcasm in larger amounts of unlabeled text is unlikely to be the same. This raises questions about the generalization of our approach when more data is used, since retrieval of texts of opposite polarity grows more likely.

6.2 Measuring model performance

Since our approach for retrieval of texts of same polarity through fine-tuned sentence embeddings has not been attempted before, there exist no qualitative metrics. We think that the suggestions we made in Section 5 are reasonable and intuitive, but there may be other metrics or weightings that could provide more insights. Additionally, comparing models trained with different loss functions is challenging due to the different types of examples they require. To address this, we selected dropout values to achieve consistent processing times across all models. Still, the training sets for TripletLoss ended up being about 10% bigger and thus also had more training time. Unfortunately, we were unable to re-run evaluations

before the submission deadline, hence this bias should be considered. However, TripletLoss models outperformed others even with limited training iterations (their metrics after only 9 iterations exceed most final scores of the other models), indicating that this difference does not invalidate our tests.

Unlike typical research on loss functions, we did not consider the loss values obtained during training or evaluation. This is because we find them to be uninformative in our context. Our goal is to balance two potentially mutually exclusive objectives: grouping sentences by polarity while preserving their semantic similarity. Since the loss only accounts for the former objective and is computed differently for each function used, we opted for the Polarity Score we introduced in Section 5 for comparability.

6.3 Hardware Limitations

Due to limited hardware availability (CPU-only laptop), most of the computations, including embedding computation and fast retrieval of nearby sentences, were performed on the free tier of Google Colab[5]. It allows the use of a GPU (NVIDIA T4 Tensor Core GPU), however only with significant time restrictions. As a result, lengthy computations were not feasible, and our experimental design involved using only a subset of available sentences and training for few iterations. Despite these limitations, we believe that general tendencies can still be observed, but more extensive experiments would be needed for a clearer picture.

6.4 Failed initial attempts

Initial research for this project involved exploring transformer architectures and generative adversarial networks (GANs) for text generation. However, due to the hardware limitations mentioned above, we were unable to train transformer architectures in a reasonable amount of time. For GANs, handling the discrete nature of texts proved challenging, and attempts using Variational Auto Encoders on our own naive sentence embeddings to circumvent these were unsuccessful. We also considered existing models for sarcasm generation of Mishra et al. and Joshi et al., however, here we encountered many issues with poorly documented code repositories and errors. As all these preliminary attempts were not directly related to the results presented in this report, we chose to exclude them.

7 Conclusion and Future work

In this paper, we investigated the potential of encoding sarcastic polarity in sentence embeddings through fine-tuning. Our experimental approach involved enhancing the 'all-miniLM-L6-v2' model using various loss functions from the sentence-transformers Python library. Despite limited hardware resources, our results demonstrated promising performance when training embeddings with TripletLoss using small margin values. However, further experiments are warranted to determine the effectiveness of our approach for a fully functional sarcasm retrieval system. Specifically, a more comprehensive investigation of different margin parameters is needed. It is worth noting that our experiments were conducted on a relatively small dataset of similarly structured sentences, and thus, empirical results on larger datasets are crucial for validating the scalability of our approach.

While our simplistic approach to sarcasm retrieval may not be sufficient for a full-fledged retrieval system due to the observed trade-off between polarity and semantic similarity, our experiments give evidence that our approach is heading in the right direction and could serve as a crucial building block for a more sophisticated and refined architecture.

The code for this project is available at https://github.com/jensbreitung/sarcasm-retrieval.

[5] https://colab.research.google.com/

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, 2006. doi:10.1109/CVPR.2006.100.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. Efficient natural language response suggestion for smart reply, 2017.

Aditya Joshi, Anoop Kunchukuttan, Pushpak Bhattacharyya, and Mark Carman. Sarcasmbot: An open-source sarcasm-generation module for chatbots. 08 2015.

Aditya Joshi, Pushpak Bhattacharyya, and Mark James Carman. Automatic sarcasm detection: A survey, 2016.

Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers, 2021.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heinz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey, 2021.

Abhijit Mishra, Tarun Tater, and Karthik Sankaranarayanan. A modular architecture for unsupervised sarcasm generation. 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)*, pages 6144–6154, Hong Kong, China, November 2019. Association for Computational Linguistics. doi:10.18653/v1/D19-1636. URL https://aclanthology.org/D19-1636.

Rishabh Misra and Prahal Arora. Sarcasm detection using news headlines dataset. *AI Open*, 4:13–18, 2023. ISSN 2666-6510. doi:https://doi.org/10.1016/j.aiopen.2023.01.001. URL https://www.sciencedirect.com/science/article/pii/S2666651023000013.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019. URL http://arxiv.org/abs/1908.10084.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015. doi:10.1109/cvpr.2015.7298682. URL https://doi.org/10.1109%2Fcvpr.2015.7298682.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Appendix A Extended results

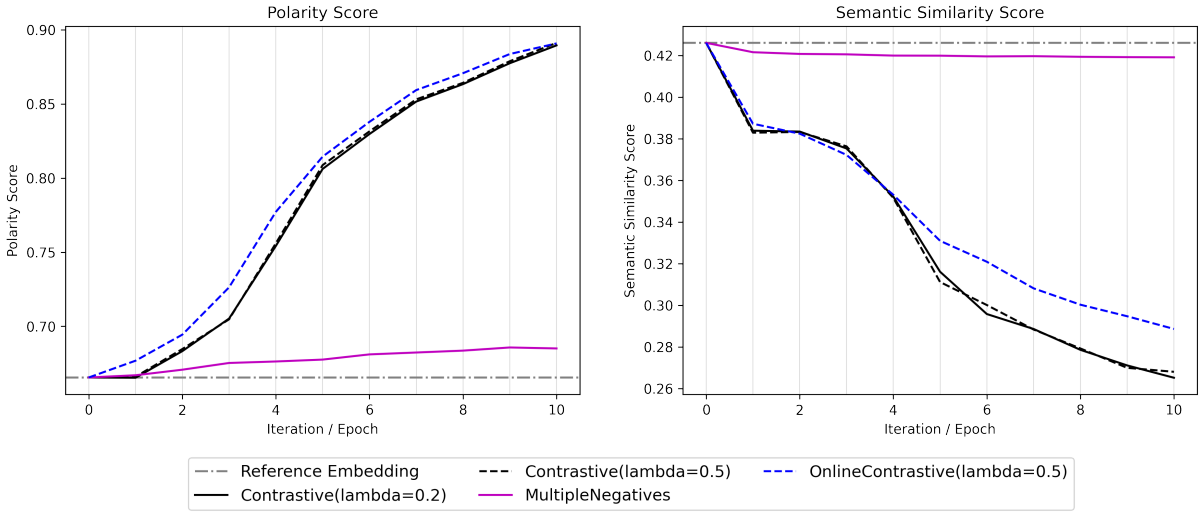


Figure 4: Polarity and semantic similarity scores for all non-triplet-based models

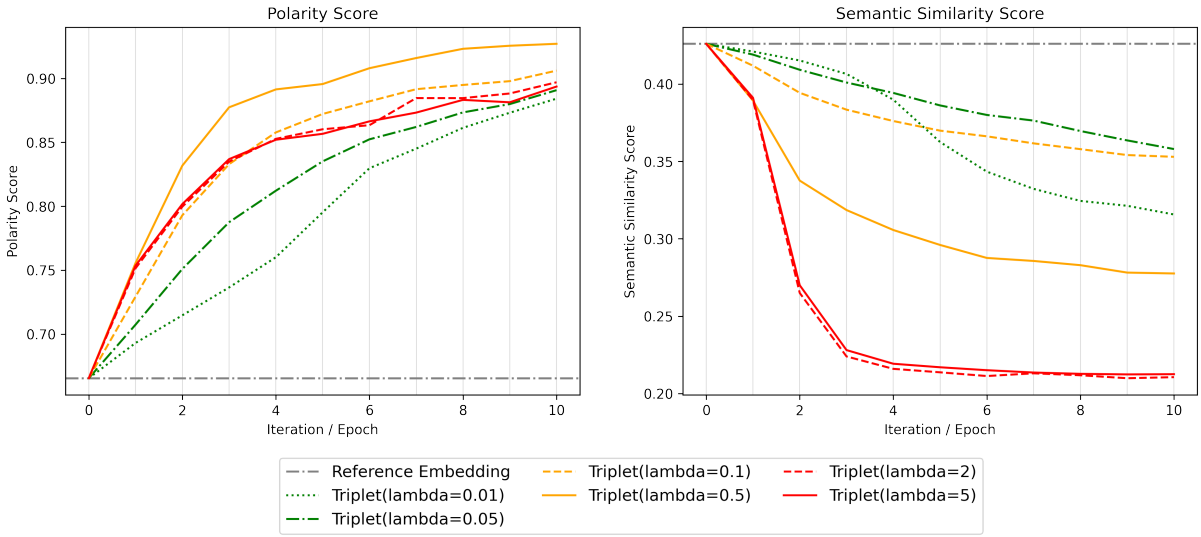


Figure 5: Polarity and semantic similarity scores for all triplet-based models

Model Name	Polarity Score after i-th iteration										
	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
Contrastive(lambda=0.2)	0.6655	0.6653	0.6832	0.7052	0.7542	0.8062	0.8297	0.8516	0.8633	0.8774	0.8895
Contrastive(lambda=0.5)	0.6655	0.6660	0.6846	0.7047	0.7560	0.8087	0.8315	0.8531	0.8642	0.8788	0.8912
MultipleNegatives	0.6655	0.6670	0.6707	0.6752	0.6763	0.6775	0.6810	0.6823	0.6836	0.6857	0.6850
OnlineContrastive(lambda=0.5)	0.6655	0.6768	0.6944	0.7263	0.7772	0.8145	0.8379	0.8593	0.8707	0.8836	0.8908
Triplet(lambda=0.01)	0.6655	0.6931	0.7147	0.7365	0.7601	0.7953	0.8298	0.8451	0.8614	0.8732	0.8842
Triplet(lambda=0.05)	0.6655	0.7074	0.7510	0.7875	0.8122	0.8351	0.8524	0.8621	0.8734	0.8799	0.8908
Triplet(lambda=0.1)	0.6655	0.7293	0.7929	0.8329	0.8578	0.8722	0.8821	0.8916	0.8949	0.8979	0.9061
Triplet(lambda=0.5)	0.6655	0.7556	0.8318	0.8774	0.8914	0.8956	0.9080	0.9160	0.9232	0.9256	0.9271
Triplet(lambda=2)	0.6655	0.7515	0.7997	0.8349	0.8527	0.8602	0.8634	0.8846	0.8846	0.8882	0.8970
Triplet(lambda=5)	0.6655	0.7536	0.8017	0.8369	0.8521	0.8567	0.8664	0.8733	0.8832	0.8813	0.8937

Table 3: Polarity scores of the different models over the course of ten iterations.

Model Name	Semantic Similarity Score after i-th iteration										
	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
Contrastive(lambda=0.2)	0.4260	0.3838	0.3834	0.3753	0.3522	0.3161	0.2957	0.2886	0.2785	0.2711	0.2651
Contrastive(lambda=0.5)	0.4260	0.3829	0.3832	0.3763	0.3516	0.3111	0.3001	0.2883	0.2792	0.2699	0.2680
MultipleNegatives	0.4260	0.4215	0.4207	0.4205	0.4199	0.4199	0.4195	0.4196	0.4193	0.4191	0.4191
OnlineContrastive(lambda=0.5)	0.4260	0.3872	0.3824	0.3722	0.3530	0.3309	0.3209	0.3081	0.3002	0.2947	0.2886
Triplet(lambda=0.01)	0.4260	0.4210	0.4152	0.4065	0.3899	0.3626	0.3434	0.3323	0.3245	0.3213	0.3157
Triplet(lambda=0.05)	0.4260	0.4190	0.4092	0.4010	0.3943	0.3862	0.3800	0.3764	0.3697	0.3636	0.3580
Triplet(lambda=0.1)	0.4260	0.4119	0.3943	0.3834	0.3761	0.3699	0.3662	0.3617	0.3579	0.3541	0.3530
Triplet(lambda=0.5)	0.4260	0.3893	0.3376	0.3186	0.3057	0.2960	0.2876	0.2857	0.2830	0.2782	0.2776
Triplet(lambda=2)	0.4260	0.3897	0.2650	0.2240	0.2160	0.2137	0.2113	0.2131	0.2118	0.2099	0.2106
Triplet(lambda=5)	0.4260	0.3911	0.2700	0.2281	0.2193	0.2170	0.2151	0.2136	0.2127	0.2124	0.2125

Table 4: Semantic similarity scores of the different models over the course of ten iterations.