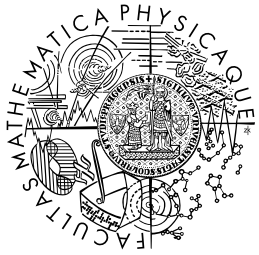

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

David Burian

Document embedding using Transformers

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Jindřich, Libovický Mgr. Ph.D.

Study programme: Computer Science

Study branch: Artificial Intelligence

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: Document embedding using Transformers

Author: David Burian

Institute: Institute of Formal and Applied Linguistics

Supervisor: Jindřich, Libovický Mgr. Ph.D., Institute of Formal and Applied Linguistics

Abstract: Abstract.

Keywords: text embedding document embedding transformers document classification document similarity

Contents

Introduction

Text embeddings are the center point of Natural Language Processing (*NLP*) in machine learning. Although it is difficult to show what information embeddings contain, we often think of them as encodings of the input's meaning. This thesis focuses on dense representations of long continuous pieces of text or simply *documents*. Document embeddings condense information into a fixed-sized vector, thus making any subsequent computation significantly faster. For some tasks, such as document semantic search or clustering, embeddings are so crucial that the computation without them would be infeasible. For other tasks, such as classification or prediction, embeddings provide meaningful features with lower dimensionality than the original text. However, training a document embedding model presents several challenges.

As texts get longer, they cover more topics, which connect in increasingly complex ways. This increase in complexity can be well illustrated with the Transformer architecture [Vaswani et al., 2017]. Transformers have become state-of-the-art architectures for many NLP tasks [Devlin et al., 2019, Liu et al., 2019], particularly for sentence embedding with models such as SBERT [Reimers and Gurevych, 2019] or SimCSE [Gao et al., 2021]. However, using Transformers for longer inputs presents a practical challenge as the Transformer's memory footprint scales quadratically with the input length. *Efficient Transformers* [Tay et al., 2022], such as Longformer [Beltagy et al., 2020], decrease the memory consumption of a vanilla Transformer architecture by replacing its self-attention layer with a less powerful counterpart. With these alterations, Transformers can be used even for tasks where large contexts are necessary, such as document summarization or embedding. However, despite the theoretical advancements, training a Transformer document embedding model still requires a large amount of computational resources.

Another challenge lies in the lack of supervised datasets due to the time-consuming and complex annotation of documents. Consequently, the training of document embeddings is either completely unsupervised or takes advantage of a structure within the corpora. Previous unsupervised training approaches for Transformers usually rely on contrastive learning, which requires either a large amount of memory [Neelakantan et al., 2022] or a complex training system, such as maintaining two embedding models [Izacard et al., 2021]. Other training approaches [Ostendorff et al., 2022, Cohan et al., 2020] focus only on document corpora with an inherent structure, such as Wikipedia articles connected via links or academic papers related via citations. However, such embedding models lack the universality of an embedding model trained on a mixture of document formats.

Ultimately, the lack of supervised corpora also hinders the evaluation of document embedding models. Supervised document datasets also suffer from low quality, where a document's label is derived automatically instead of being directly based on the document's content. Furthermore, while datasets may claim to contain documents, they often consist of shorter texts, such as abstracts. This results in a small number of available high-quality datasets that evaluate embeddings on long pieces of text. However, a document embedding model should be

tested on several tasks spanning different lengths, formats, and topics in order to show consistent performance.

In this work, we tackle some of the challenges described above while coping with the rest. We train a Transformer-based document embedding model with a small number of resources on two fully unsupervised text corpora without any structure. We evaluate our model on several tasks that cover sentiment analysis, citation prediction, and semantic search.

We base our method on teacher-student training, where a student model is trained to mimic the teacher model. However, instead of one teacher model, we use two and capitalize on both of their strengths. Each teacher model is a document embedding model with a different architecture generating embeddings with distinct qualities. We distill both of these qualities into a single student model. For the student model, we use an efficient Transformer with sparse attention that can embed texts up to 4096 tokens long.

1. Document representation

Representing a piece of text by a dense vector, also known as text embedding, is ubiquitous in Natural Language Processing (*NLP*). However, embedding long, continuous pieces of text such as documents is substantially more complex than embedding words or sentences, as the longer and more involved input still needs to be compressed into a similarly sized vector. In this chapter, we first briefly explore the use cases of document embeddings. Afterward, we describe the qualities of document embeddings that we deem beneficial and build up the motivation behind our training method, which we describe in Chapter 3.

1.1 Use cases of document embeddings

Embeddings that capture the document's semantics in a low-dimensional vector reduce the noise in the raw input while making the subsequent operations more efficient. These are the main reasons why document embeddings are widely used across different tasks such as classification [Cohan et al., 2020, Neelakantan et al., 2022, Izacard et al., 2021, Ostendorff et al., 2022], ad-hoc search [Singh et al., 2022, Zamani et al., 2018], query answering [Neelakantan et al., 2022], visualization [Cohan et al., 2020, Dai et al., 2015] and regression [Singh et al., 2022].

While some models [Singh et al., 2022] can generate different embeddings for a single input depending on the task, most embedding models output a single vector that is effective across many types of tasks [Neelakantan et al., 2022, Cohan et al., 2020, Ostendorff et al., 2022]. This shows that embedding models can substitute several dedicated models, severely saving time and resources.

1.2 Desirable qualities of embeddings

The usefulness of document embeddings stems from 3 properties. An ideal document embedding (1) represents the document's text faithfully (2) with a single vector (3) of low dimension.

In this section, we focus on faithful document representation, which we view as a composition of two abstract qualities: *structural* and *contextual*. Document embedding with structural quality (or structural document embedding) faithfully models the relationships between words or word sequences. Based on these relationships, the embedding can capture meaning even for documents with complex structures. On the other hand, contextual document embedding composes the meaning of all processed words, capturing the overall theme or topic of the document. We view these qualities as scales, so a document embedding may have high structural but low contextual quality. Such embedding captures the relationships between words very well and thus faithfully represents the meaning of sections with unambiguous context. However, the embedding can easily misinterpret sections where context is needed to disambiguate between several meanings.

Since each document embedding is produced by a model, we may attribute similar qualities to the models themselves. In this sense, we speak of the model's

structural or *contextual capacity*.

In the following subsections, we focus on each quality separately, describing each in more detail. At the end of this section, we compare the two qualities and outline our proposed training method described in Chapter 3.

1.2.1 Structural quality of document embeddings

Structural quality defines how well the embedding captures relationships within the input text. The more complex the relationship is, the higher structural quality is needed to interpret the text correctly. For instance, we list exemplary observations based on word relationships in a sentence: "Fabian likes playing the guitar, but Rebecca does not.":

Observation 1. "Fabian" likes something based on the words "Fabian likes"

Observation 2. A guitar can be played based on the words "playing the guitar"

Observation 3. The two sequences of words separated by a comma are in opposition based on the words ", but"

Observation 4. "Fabian" likes to play the guitar based on Observations 1 and 2.

The relationships get more and more complex as the number of participating words increases (Observations 1-3) or as we layer the relationships (Observation 4). Therefore, an embedding would need an increasing level of structural quality to capture Observations 1-4 correctly.

Embedding can reflect world relationships only if the model that produced it compares the participating words to each other. Based on the number and complexity of comparisons the model makes, we can derive its level of structural capacity. A good example of a model with high structural capacity is Transformer [Vaswani et al., 2017]. Transformer's self-attention layer allows each word to exchange information with other words. Additionally, self-attention allows the aggregation of several words into one. Thanks to Transformer's layered architecture, such aggregations can be compared similarly on higher levels. An example of a model with low structural capacity is Paragraph Vector [Le and Mikolov, 2014]. Paragraph Vector compares words only in a single fully connected layer. Such architecture prevents the understanding of more complex relationships that build on other relationships, such as Observation 4.

1.2.2 Contextual quality of document embeddings

The contextual quality of a document embedding defines how well the embedding captures the overall meaning of longer texts. The longer the sequence, the higher the contextual quality of an embedding correctly capturing its overall topic. For instance, let us consider two documents: 1. a description of a typical commercial turbo-jet airplane and 2. a recipe for spicy fried chicken wings. A document embedding with high enough contextual quality would reflect that the following sentence's meaning: "Left wing is too hot." dramatically differs between the two documents and would accordingly adjust the sentence's contribution to the resulting document embedding.

Provided the document's text is cohesive and continuous, capturing its overall meaning gets easier as the text's length increases. Intuitively, the more words we see, the more information we know about their common theme. As the theme becomes increasingly more refined, fewer meanings that correspond to it. Consequently, we judge a model's contextual capacity based on the maximum length of an input that the model can to process. This number is also commonly known as the maximum context length of a model. An example of a model with good contextual capacity is Paragraph Vector [Le and Mikolov, 2014], which can process, in theory, indefinitely long sequences[1]. Additionally, Paragraph Vector stores a single vector per document, which is iteratively compared to all words within it. This allows the model to adjust individual words' contribution to the document's meaning. On the other hand, Transformer [Vaswani et al., 2017] has much smaller contextual capacity as its memory requirements grow quadratically with the length of the input, which in practice significantly shortens Transformer's maximum context length.

1.2.3 Combining structural and contextual qualities

Each quality describes a different aspect of faithful representation. Structural quality is focused more on local relationships of words, while contextual quality considers mainly the global picture. From a performance standpoint, structural quality is oriented more toward precision, while contextual quality is oriented more toward recall. In a way, the two qualities complement each other. Contextual quality brings in the overall document theme, while structural quality brings in the detailed meaning of a shorter sequence. We hypothesize that these two pieces of information can be aligned to produce precise, unambiguous document embedding that outperforms embeddings with just a single quality.

While we predict that a mix of both qualities is beneficial, we are unsure which ratio would be the most performant. Arguably, structural quality is more important than contextual since, in extreme cases, it can model relationships so complex, that they span the entire document, substituting contextual quality's role. On the other hand, we can expect that, for a given input length, an embedding model with high structural capacity will be larger than an embedding model with high contextual capacity. The reason is that the number of total relationships found in a document grows exponentially with the length of the document, whereas the number of topics covered can grow only linearly.

Our training method stems from these observations and hypotheses. We align the two qualities with each other and find the ideal ratio of the two qualities that produce the best-performing embeddings. We describe our training method in detail in Chapter 3.

[1] Provided the vocabulary size stays constant.

2. Related Work

This chapter reviews the research that we consider relevant to embedding long texts using transformers [Vaswani et al., 2017]. First, we summarize efforts that have gone into making transformers more efficient to so that they can process long inputs. These advancements are crucial to embedding documents, often much longer than the standard 512 tokens. In the following section, we describe approaches to training embedding models.

2.1 Efficient transformers

Though the transformer has proven to be a performant architecture in the world of NLP [Devlin et al., 2019, Liu et al., 2019, Reimers and Gurevych, 2019], it has one inherent disadvantage regarding longer texts. The self-attention layer, the principal part of the transformer, consumes a quadratic amount of memory in the length of the input. This significantly limits the transformer's applicability in tasks that require longer contexts such as document retrieval or summarization.

Thanks to the popularity of the transformer architecture, a large amount of research is focused on making transformers more efficient [Tay et al., 2022]. Most of these efforts fall into one of the following categories:

1. Designing a new memory-efficient attention mechanism
2. Using a custom attention implementation
3. Combining the transformer with another architecture

We review each category separately, though these approaches can be combined [Child et al., 2019, Beltagy et al., 2020]. In the section dedicated to custom implementation of self-attention, we also mention commonly used implementation strategies that make transformers more efficient in practice.

2.1.1 Efficient self-attention mechanisms

The classical scaled dot-product self-attention [Vaswani et al., 2017] is the most resource-intensive component of the transformer. The core of the problem is the multiplication of $N \times d$ query matrix and $N \times d$ key matrix, where N is the input length, and d is the dimensionality of the self-attention layer. Efficient attention mechanisms approximate this multiplication, avoiding computing and storing the $N \times N$ resulting matrix.

Sparse attention

Sparse attention approximates full attention by ignoring dot products between some query and key vectors. Though it may seem like a crude approximation, research shows that the full attention focuses mainly on a few query-key vector combinations. For instance, Kovaleva et al. [2019] showed that full attentions exhibit only a few repeated patterns, and by disabling some attention heads, we

can increase the model's performance. These findings suggest that full attention is over-parametrized, and its pruning may be beneficial. Moreover, Child et al. [2019] showed that when processing images, the attention is composed of repeated sparse patterns and that by approximating full attention using such sparse patterns, we can increase the model's efficiency without sacrificing performance.

Sparse attentions typically compose several attention patterns. One of these patterns is often full attention limited only to a neighborhood of considered token. This pattern corresponds to the findings of Clark et al. [2019], who found that full attention focuses a lot of on previous and next tokens. Another sparse attention pattern is usually dedicated to enabling a broader exchange of information between tokens. In Sparse Transformer [Child et al., 2019], distant tokens are connected by several pre-selected tokens uniformly distributed throughout the input. In Longformer [Beltagy et al., 2020], every token can attend to every kth distant token to increase its field of vision. BigBird [Zaheer et al., 2020] computes dot products between randomly chosen pairs of key-query vectors. These serve as connecting nodes for other tokens exchanging information. The last typical sparse attention pattern is a global attention that is computed only on a few tokens. Though such attention pattern is costly it is essential for tasks which require a representation of the whole input [Beltagy et al., 2020]. In Longformer, some significant input tokens, such as the `[CLS]` token, attend to all other tokens and vice-versa. BigBird computes global attention also on a few extra tokens added to the input.

Sparse attention patterns do not have to be fixed but can also change throughout the training. Sukhbaatar et al. [2019] train a transformer that learns optimal attention span. In their experiments, most heads learn to attend only to a few neighboring tokens, which makes the model more efficient. Reformer [Kitaev et al., 2020] computes the full self-attention only between close key and query tokens while letting the model decide which two tokens are "close" and which are not. That enables the model to learn optimal attention patterns between tokens to a certain degree.

Low-rank approximations and kernel methods

Besides using sparse attention, other techniques make self-attention more efficient in memory and time. Wang et al. [2020] show that the attention matrix $A := \text{softmax}(\frac{QK^T}{d})$ is of low rank and that it can be approximated in fewer dimensions. By projecting the $N \times d$-dimensional key and value matrices into $k \times d$ matrices, where $k \ll N$, they avoid the expensive $N \times N$ matrix multiplication. The authors show that the empirical performance of their model is on par with the standard transformer models such as RoBERTa [Liu et al., 2019] or BERT [Devlin et al., 2019].

In another effort, Choromanski et al. [2020] look at the standard softmax self-attention through the lens of kernels. The authors use feature engineering and kernels to approximate the elements of the previously mentioned attention matrix A as dot products of query and key feature vectors. Self-attention can then be approximated as a multiplication of four matrices: the projected query and key matrices, the normalization matrix substituting the division by d, and the value matrix. That allows the matrix multiplications to be reordered, first multiplying

the projected key and the value matrix and then multiplying by the projected query matrix. Such reordering saves time and space by a factor of $O(N)$ making the self-attention linear in input length.

2.1.2 Implementation enhancements

Transformer models can be made more efficient through a purposeful implementation. As modern hardware gets faster and has more memory, implementation enhancements can render theoretical advancements such as sparse attention unnecessary. For example, Xiong et al. [2023] train a 70B model on sequences up to 32K tokens with full self-attention. Nevertheless, the necessary hardware to train such models is still unavailable to many; therefore, there is still the need to use theoretical advancements together with an optimized implementation. For instance Jiang et al. [2023] trained an efficient transformer that uses sparse attention and its optimized implementation. The resulting model beats competitive models with twice as many parameters in several benchmarks.

Optimized self-attention implementation

Efficient self-attention implementations view the operation as a whole rather than a series of matrix multiplications. That enables optimizations that would not be otherwise possible. The result is a single GPU kernel that accepts the query, key, and value vectors and outputs the result of a standard full-attention. Rabe and Staats [2021] proposed an implementation of full self-attention in the Jax library[1] for TPUs that uses logarithmic amount of memory in the length of the input. Dao et al. [2022] introduced Flash Attention, which focuses on optimizing IO reads and writes and achieves non-trivial speedups. Flash Attention offers custom CUDA kernels for both block-sparse and full self-attentions. Later, Dao [2023] improved Flash Attention's parallelization and increased its efficiency even more. Though using optimized kernel is more involved than spelling the operations out, libraries like xFormers[2] and recent versions of PyTorch[3] make it much more straightforward. Unfortunately, as of this writing, only xFormers support custom masking in self-attention.

Mixed precision, gradient checkpointing and accumulation

Besides the above-mentioned recent implementation enhancements, some techniques have been used not just in conjunction with transformers. We mention them here, mainly for completeness, since they dramatically lower the required memory of a transformer model and thus allow training with longer sequences.

Micikevicius et al. [2017] introduced mixed precision training, which almost halves the memory requirements of the model as almost all of the activations and the gradients are computed in half precision. As the authors show, with additional techniques such as loss scaling, mixed precision does not worsen the results compared to traditional single precision training. In another effort to lower the

[1] `https://github.com/google/jax`

[2] `https://github.com/facebookresearch/xformers`

[3] `https://pytorch.org/docs/2.2/generated/torch.nn.functional.scaled_dot_product_attention.html`

memory required to train a model, Chen et al. [2016] introduced gradient checkpointing to trade speed for memory. With gradient checkpointing activations, some layers are dropped or overwritten to save memory but then need to be recomputed again during backward pass. Another popular technique is gradient accumulation, which may effectively increase batch size while maintaining the same memory footprint. With gradient accumulation, gradients are not applied immediately but are accumulated for k batches and only then applied to the weights. That has a similar effect as multiplying the batch size by k but is not equivalent since operations like Batch Normalization [Ioffe and Szegedy, 2015] or methods such as in-batch negatives behave differently. Nevertheless, gradient accumulation is a good alternative, especially if the desired batch size cannot fit into the GPU memory.

2.1.3 Combination of model architectures

In order to circumvent the problem of the memory-intensive self-attention layer, some research efforts explored combining the transformer architecture with another architectural concept, namely recursive and hierarchical networks. The typical approach of these models is not to modify the self-attention or the maximum length of input the transformer can process but instead to use transformers to process smaller text segments separately and contextualize them later. Dai et al. [2019] proposes using a recursive architecture of transformer nodes, where each transformer receives the hidden states of the previous one. Since gradients do not travel between the nodes, processing longer sequences requires only constant memory. The resulting model achieves state-of-the-art performance on language modeling tasks with a parameter count comparable with the competition. Yang et al. [2020] use a simpler architecture of a hierarchical transformer model. First, transformers individually process text segments, producing segment-level representations, which are fed to another document-level transformer, together with their position embeddings. The authors pre-train with both word-masking and segment-masking losses. After finetuning it on the target tasks, the model beats scores previously set by recurrent networks.

2.2 Training document embedding models

When we train a document embedding model, we aim to improve the performance of the model's embeddings on downstream tasks. There are many types of downstream tasks, such as classification, retrieval, clustering, or visualizations, and an embedding model is generally expected to perform well in all of them. Therefore, there is no clear optimization objective, nor is there an objective universally agreed upon to outperform others. That makes the task of training document embedding models diverse. All training techniques, however, have to adapt to the currently available training document corpora. Due to higher annotation costs and complexity, there are fewer supervised corpora of documents than supervised corpora of shorter sequences such as sentences. Nevertheless, some datasets offer rich metadata useful for constructing supervised datasets. A typical example is the Semantic Scholar corpus [Ammar et al., 2018] that links academic papers via citations.

In the simplest case, embedding models are trained only through word or token prediction. Paragraph Vector [Le and Mikolov, 2014] is trained on the prediction of the masked-out words given the embeddings of the surrounding words and the embedding of the given document. However, transformers cannot learn document embedding through token prediction. So, a transformer-based embedding model is typically trained using pre-training on large unlabeled corpora and then finetuned. In pre-training, the model gains most of its knowledge, and then, in the finetuning phase, the model improves the quality of its embeddings. For instance, Cohan et al. [2020], Izacard et al. [2021] warm start their embedding models from SciBERT [Beltagy et al., 2019] and BERT [Devlin et al., 2019], both trained using Masked Language Modelling (MLM) on an unsupervised text corpus. However, these models differ in how they are finetuned.

Cohan et al. [2020] use a triplet loss that takes three documents: a query, a positive, and a negative document. Triplet loss then minimizes the distance between the query and the positive document while maximizing the distance between the query and the negative document. To obtain a positive and a negative document for a given query document, the authors leverage the structure of Semantic Scholar corpus [Ammar et al., 2018]. Ostendorff et al. [2022] repeats the experiment but shows a more elaborate method of sampling negative papers improves the final model.

Another popular technique is to train the model by contrasting several inputs' embeddings against each other. For each input, there is at least one similar to it (positive), while the others are usually deemed as dissimilar (negative). The loss would then minimize cross-entropy between the true similarities and those computed from the input's embeddings. As with the triplet loss, the main difference between models is how they obtain the positive and negative documents. Neelakantan et al. [2022] use the given input as a positive, while all other inputs in the batch are considered negatives. Using in-batch negatives is very efficient since the model can utilize each input once as a positive and several times as a negative. As the authors point out, the key to this technique is to have large batch sizes – the authors suggest batch sizes of several thousand documents. Izacard et al. [2021] obtain positives by augmenting the original document. In contrast to the previously mentioned model, the authors use negatives computed in previous batches. While using out-of-batch negatives avoids the need for a large batch size, a set of new problems surfaces, such as making sure that the trained model does not change too quickly, which would make the stored negatives irrelevant and possibly harmful to the training. The authors solve this issue by a secondary network, whose parameters are updated according to the primary embedding model.

An embedding model usually only outputs one embedding for a single input, yet some models can generate more embeddings for a given input depending on external factors. Singh et al. [2022] train an embedding model whose embeddings are finetuned for a given type of downstream task. The model's task-specific modules are trained end-to-end on supervised data collected by the authors. The proposed model can share knowledge across all types of tasks thanks to the use of control codes and a layer connecting all task-specific modules. While the idea seems reasonable, the authors achieve only a fractional improvement over state-of-the-art models that generate a single embedding for a single input.

2.2.1 Comparison to the proposed training method

Like other transformer-based embedding models, ours is warm-started from a pre-trained model. However, the following finetuning of our embeddings avoids some of the downsides of the previously mentioned methods. First, it does not require any structure in the training data, which is essential as there is only a limited amount of structured corpora of documents. Typically, these would be scientific papers or Wikipedia articles connected via citations or links. Our training method allows using any document corpora, such as a set of books or news articles. Secondly, it does not require large batch sizes. Despite the advancements mentioned in Section 2.1, using a consumer-grade GPU card to train a transformer-based embedding model with long inputs can still pose a practical challenge. Using a batch size of several thousand documents is unimaginable in this context. Third, our method does not require maintaining any secondary network while training the model. Though our method uses embeddings of other models, these can be generated beforehand and thus do not take up the resources needed to train the embedding model. Fourth, we aim to obtain a model usable with any continuous text. We do not limit our embedding model only to a specific field, such as scientific literature. Finally, our model generates a single embedding, which we evaluate on a diverse set of tasks, including classification of individual documents, classification of document pairs, and document retrieval.

3. Distilling qualities of document embeddings

In this chapter, we introduce our method of training document embeddings. We base our approach on teacher-student training, distilling the knowledge of two embedding models (referred to as *teachers*) into one *student* model. Section 3.1 explains our training method in detail and outlines the used loss function. In the rest of this chapter, we describe the two teacher models in Section 3.2 and the student in Section 3.3.

3.1 Training methodology

Our training methodology aims to train an embedding model such that its embeddings more faithfully represent the input. As we describe in Chapter 1, we distinguish two qualities of faithful representations: structural and contextual. The goal is to instill both qualities into a single embedding model. To do so, we use teacher-student training with two teacher embedding models, one with high structural capacity and the other with high contextual capacity.

In the following subsections, we describe teacher-student training in detail and give a high-level overview of the proposed loss function.

3.1.1 Teacher-student training

In the teacher-student training, we train a student model based on a frozen teacher model. The goal is to make the student model imitate the teacher model, thereby digesting the teacher's understanding of the input. Although the student model is generally not expected to outperform the teacher model, teacher-student training is still valuable in several situations. For instance, Sanh et al. [2019] use teacher-student training to make a model smaller while sacrificing only a fraction of its performance. In another scenario, Reimers and Gurevych [2020] use teacher-student training to enforce similarity between models' outputs, thereby giving the student model a powerful training signal.

We assume two embedding models in our setting: a structural teacher $\mathcal{T}_S$ and a contextual teacher $\mathcal{T}_C$ with high structural and contextual capacities, respectively. Teacher-student training allows us to instill both capacities into a third student model $\mathcal{S}$ while avoiding the architectural limitations of the teachers. We hypothesize that we can efficiently direct the two training signals so that they do not push against each other.

3.1.2 Abstract loss formulation

We instill a quality of a teacher's embedding by simply enforcing a similarity between the teacher's and student's embedding. Since we have two teachers, we use two similarities $\mathcal{L}_S$, and $\mathcal{L}_C$, which compare the student's embedding $y_{\mathcal{S}}$ with the structural teacher's embedding $y_{\mathcal{T}_S}$ and the contextual teacher's embedding

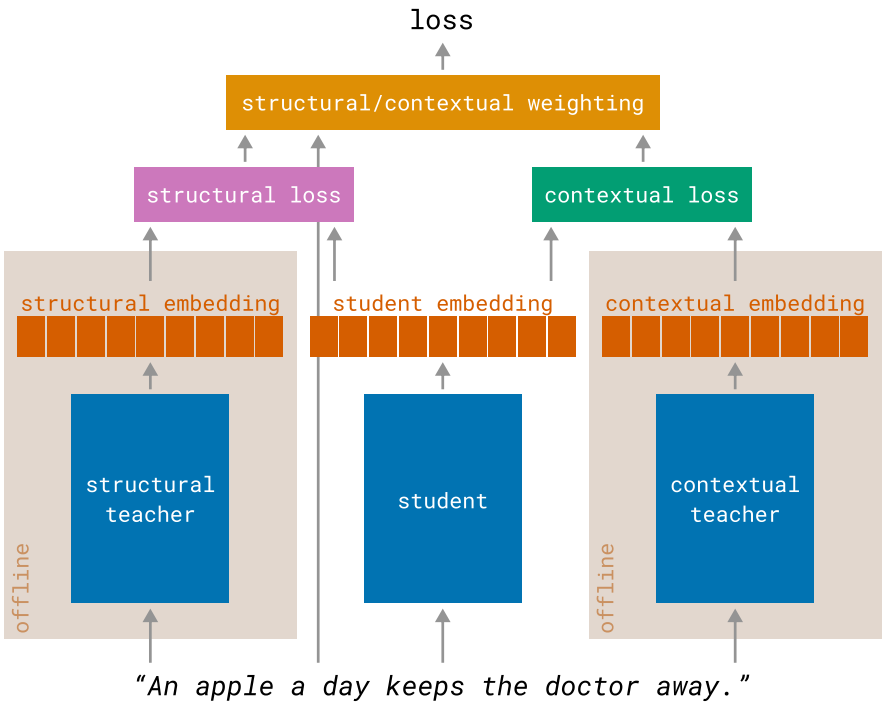

Figure 3.1: The architecture of our teacher-student training. We distill the qualities of the teachers' embeddings through corresponding losses into a student model. Since we do not update the weights of either teacher, the generation of their embeddings can be done offline before training.

$y_{\mathcal{T}_C}$, respectively. We show a graphical overview of the training architecture in Figure 3.1.

To regulate the mixture of $\mathcal{L}_S$ and $\mathcal{L}_C$, we introduce weighting parameter λ. In the most general form, we assume λ to be dependent on the input text x since the performance of the teacher models might vary across different inputs. In particular, we can expect λ to depend on the length of the input since, for shorter inputs, the context is minimal and, therefore, expendable. Abstract formulation of the loss is given in Equation 3.1. We explore concrete options for $\mathcal{L}_S$, $\mathcal{L}_C$ and $\lambda(x)$ in Chapter 4.

$$\mathcal{L}(x, y_\mathcal{S}, y_{\mathcal{T}_S}, y_{\mathcal{T}_C}, \lambda) = \lambda(x)\mathcal{L}_S(y_\mathcal{S}, y_{\mathcal{T}_S}) + (1 - \lambda(x))\mathcal{L}_C(y_\mathcal{S}, y_{\mathcal{T}_S}) \tag{3.1}$$

The two losses could push against each other and slow down or halt the training. To avoid that, we choose one of the losses to be more strict while the other to be more forgiving. In that way, the more forgiving loss should adapt to the strict one instead of pushing against it. As mentioned in Section 1.2.3, we view structural quality as the more important. Therefore, we choose the structural loss $\mathcal{L}_S$ as the stricter and exact loss, forcing the student to mimic the structural teacher as much as possible. On the other hand, the contextual loss $\mathcal{L}_C$ should give the student model more freedom in the form of the produced embedding but still force it to incorporate the information from the contextual embedding.

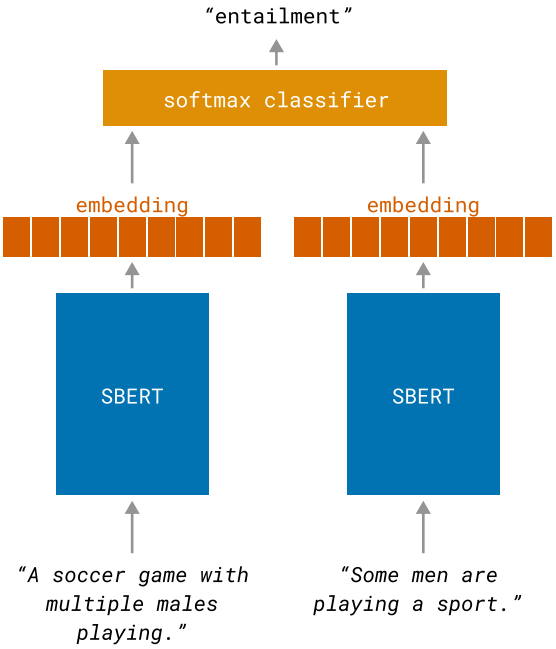

Figure 3.2: Siamese network architecture used to train SBERT. The pair of sentences from an NLI dataset is classified into three classes "entailment", "neutral" and "contradiction".

3.2 Teacher models

This section introduces the teacher models used during our experiments in Chapter 4. We chose Sentence-BERT [Reimers and Gurevych, 2019] as the structural teacher model and Paragraph Vector [Le and Mikolov, 2014] (or PV) as the context teacher model. As explained in Chapter 1, each of the two mentioned models specializes in a different quality of produced embeddings. SBERT can compare word relationships on many levels and thus understand even complex text structures. However, it cannot process long texts. On the other hand, Paragraph Vectors can produce embeddings even for long documents, but they process text very shallowly, which prohibits understanding any complex structures. We hope to synthesize both qualities by combining the two teacher models in a single model.

3.2.1 SBERT

Sentence-BERT is a composition of a BERT-like [Devlin et al., 2019] encoder with a mean pooling layer above its last layer's hidden states. The model is finetuned with Natural Language Inference (NLI) datasets to produce semantically meaningful embeddings. We illustrate SBERT's training architecture in Figure 3.2. We have chosen SBERT as a structural teacher for its high structural capacity and strong performance in sentence-level text-understanding tasks [Reimers and Gurevych, 2019].

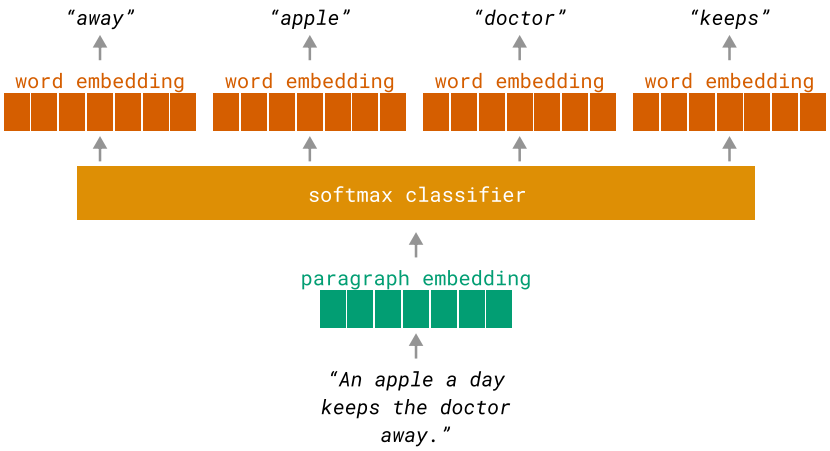

Figure 3.3: Architecture of Distributed Bag of Words. The model predicts words from a document, only using the document's embedding.

3.2.2 Paragraph Vector

Paragraph Vector [Le and Mikolov, 2014], sometimes referred to as Doc2Vec, is a simple text-embedding model that views the input as a Bag of Words (or BoW). Paragraph Vector comprises two sub-models: Distributed Memory (DM) and Distributed Bag of Words (DBOW). While each model is trained separately, the authors recommend combining both architectures into a single model, where the combined models' embeddings are simply concatenated. The models are trained to predict a word within a window in the given document. As shown in Figure 3.3, DBOW bases its prediction only on the whole paragraph's embedding. On the other hand, DM, whose architecture is depicted in Figure 3.4, additionally uses the embeddings of the surrounding words within a given window.

We chose Paragraph Vector as a contextual teacher due to its unique architecture, which forces the model to develop a single vector that summarizes the common theme of the document. Moreover, Paragraph Vector does not have a limited maximum input length, so as a contextual teacher, it will always provide some signal to the student regarding the document's context. Also, even though Paragraph Vector cannot match the performance of substantially more complex models such as Transformers, Dai et al. [2015] show that for larger datasets, Paragraph Vector, outperforms classical embedding models such as Latent Dirichlet Allocation [Blei et al., 2003] or TF-IDF weighted BoW model [Harris, 1954]. Finally, Paragraph Vector's simple architecture allows it to train on significantly larger text corpora than other bigger models, such as SBERT. Therefore, for a given computational budget, Paragraph Vector would see more documents during training than SBERT, which may give it a slight advantage.

3.3 Student model

In our teacher-student training, the student model is our primary embedding model, which we train based on the outputs of the teacher models. By using teacher-student training, we can avoid some of the architectural drawbacks of both teacher models while still benefiting from the qualities of the teachers' em-

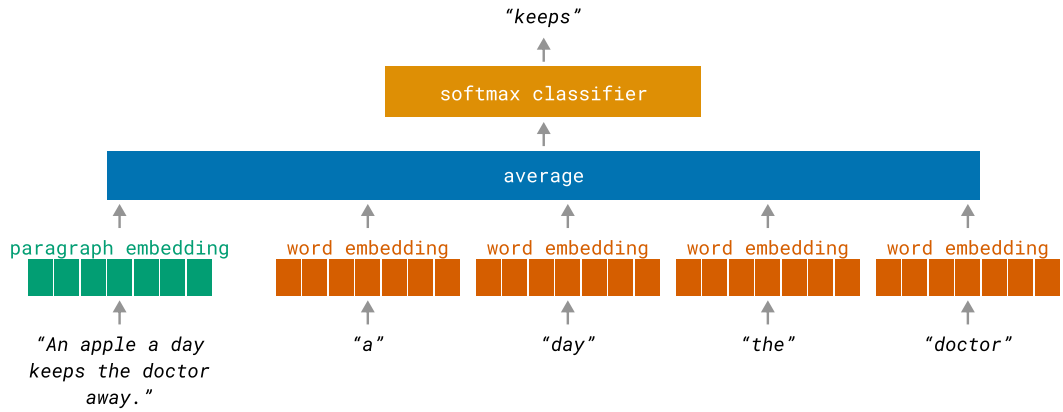

Figure 3.4: Distributed Memory model architecture. The model predicts the input words' neighboring word for the input paragraph.

beddings. We choose the student's architecture at a midpoint between the two teachers' architectures. In other words, not to be as complex as the architecture of SBERT, so it can process longer inputs while still having a manageable memory footprint, but not as simple as the architecture of PV, so that the student can model complex world relationships. We chose a Transformer with a sparse attention mechanism. Transformer is a well-tested architecture that is used throughout NLP. Additionally, with sparse attention, Transformers consume a relatively small amount of memory even for longer inputs, as we explain in Section 2.1.1. Another consideration is that we need a pre-trained model, as our method is not suited to train a model from scratch but to finetune a model's document embeddings.

Contrary to the selection of architecture, selecting a concrete model is not crucial to our method. Our choice of the concrete model is governed more by practical considerations rather than the conditions of our method. Since we have limited computational resources, we prefer a smaller model that we can fit on a consumer-grade GPU card. We also value the model's performance, ease of use, and simplicity. We choose Longformer [Beltagy et al., 2020] as it is reasonably small, memory efficient, performs above average compared to other similar models [Tay et al., 2020], and its self-attention mechanism is straightforward. Other alternatives are BigBird [Zaheer et al., 2020], or if we would not mind a more complex model, we could use Reformer [Kitaev et al., 2020], Linformer [Wang et al., 2020] or Performer [Choromanski et al., 2020].

3.3.1 Longformer

Longformer [Beltagy et al., 2020] is a Transformer encoder with sparse attention. Because we refer to Longformer's configuration and training in the following chapters, we briefly explain Longformer's self-attention mechanism and the training data of the pre-trained checkpoint.

Self-attention mechanism

Longformer has a sparse self-attention mechanism that composes three different patterns: local, global, and dilated local attention. Local attention is simply full attention but only within the neighborhood of $\frac{1}{2}\omega$ tokens on either side of the

key token. ω can be set differently per self-attention layer. In global attention, few selected tokens attend to all other tokens. The tokens on which Longformer computes global attention can be selected per each input. Global attention's parameters are not pre-trained. Instead, at the beginning of the training, they are initialized by the parameters from local attention and finetuned for a given task. With dilated local attention, every key token attends to every k neighboring query token. So, it is analogous to a one-dimensional Convolution layer [Van Den Oord et al., 2016] with stride, or dilatation of k. However, to use dilated local attention, one has to use a custom CUDA kernel or a slow implementation in Python using loops. The authors also provide a reasonably fast, memory-efficient block implementation for global and local attention.

Training

Longformer is warm-started from a RoBERTa [Liu et al., 2019] checkpoint with its learned positional embeddings duplicated eight times to support inputs up to 4096 tokens long. The authors show that duplicating RoBERTa's positional embeddings is faster than training position embeddings for all 4096 positions from scratch. Then, the authors train Longformer using MLM on long documents for 65k gradient steps to improve its capabilities for longer inputs. The training corpus overlaps with RoBERTa's pretraining corpus but is more focused on longer pieces of text. It includes the following datasets:

- Book corpus [Zhu et al., 2015]
- English Wikipedia
- One-third of articles from Realnews dataset [Zellers et al., 2019] with more than 1200 tokens
- One-third of the Stories corpus [Trinh and Le, 2018]

Unfortunately, as of this writing, the Book corpus is unavailable due to licensing issues. Moreover, we have not been able to find a comparable alternative. The Stories corpus is also unavailable. The only alternative we have found is hosted on HuggingFace[1], which, despite its description, does not seem to mimic the original dataset, since the articles it contains are extremely short. The mean word count per document is only 71 words and 99% of documents have less than 145 words.

[1] `https://huggingface.co/datasets/spacemanidol/cc-stories`

4. Experiments

In this chapter, we experiment with the training method we introduced in Chapter 3. While the main goal is to outperform both teacher models and the base student checkpoint, we also want to show how each teacher contributes to the student's performance.

This chapter is laid out as follows. We describe the training data in Section 4.1. Next, we discuss how we compare models in Section 4.2 and present the student's configuration and define the baselines in Section 4.3. Then, we experiment with the structural loss in Section 4.4. With the structural loss already given, we find the best performing contextual loss and the weighting of the two losses in Section 4.5. Finally, we summarize our experiments and findings in Section 4.6.

4.1 Training data

Our training dataset mirrors Longformer's training dataset, except for few exceptions. We leave out Book corpus [Zhu et al., 2015] and the Stories corpus [Trinh and Le, 2018] as they are currently unavailable. So, we equally sample documents from English Wikipedia and RealNews articles [Zellers et al., 2019], which are at least 1200 Longformer tokens long. We label the resulting dataset as VAL-500K and show its statistics in Table 4.1. We compile our training dataset from Longformer's training data so that the comparison between our trained student model and Longformer is more fair. In this way, the trained student model, does not see any new data compared to Longformer, and therefore any difference between the models' performances can be attributed to our training method. For the same reasons, we use identical method to generate dataset TRAIN-1M for the final training of our student model in Chapter 5.

Very similar to TRAIN-1M, VAL-500K contains long documents that are, on average, over 1300 tokens long. Consequently, only about 34% of the documents could be processed whole using a traditional Transformer such as RoBERTa [Liu et al., 2019] or SBERT. We also display the documents' length distribution in Figure 4.1. As Wikipedia contains relatively short documents, while RealNews does not contain documents shorter than 1200 tokens, the source's distributions are well-spaced.

Split	Train	Validation
Documents	500 000	10 000
Tokens	6.85e+08	1.37e+07
Tokens per document	1371±1723	1372±1717
SBERT tokens over 384	71%	70%
SBERT tokens over 512	66%	66%

Table 4.1: Statistics of VAL-500K. Apart from document count, token count, and mean token count per document, we also show the percentage of documents with the number of SBERT tokens above a given threshold.

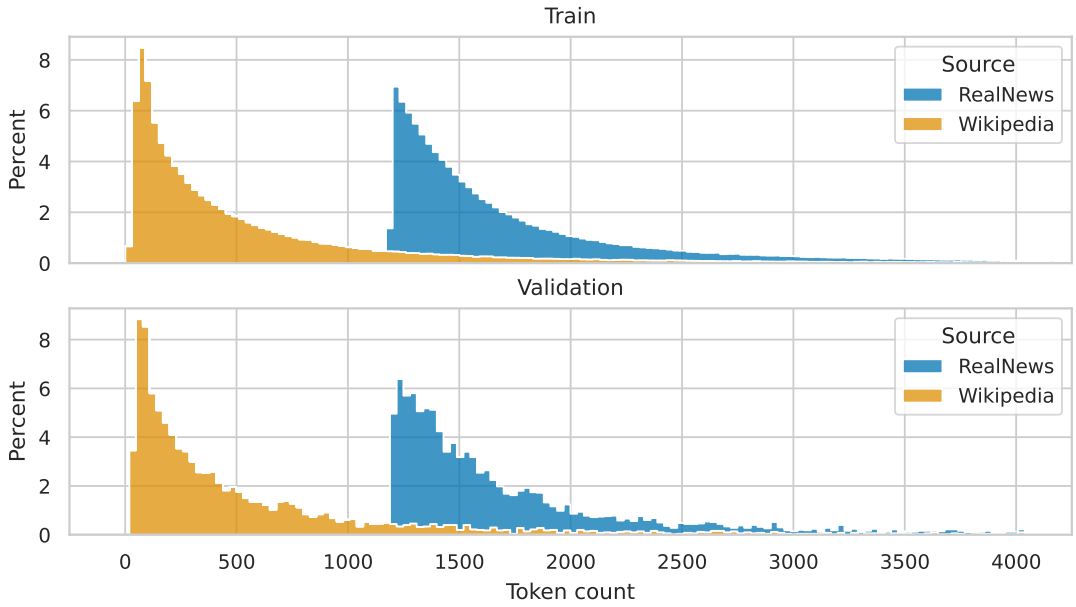

Figure 4.1: Distribution of train and validation documents' lengths for VAL-500K.

4.2 Validation tasks

We compare the embedding models based on their performance on downstream tasks. We use a subset of our evaluation tasks, described in detail in Chapter 5. We include tasks with either a large enough training split suitable for cross-validation or a validation split. As a result we validate embedding models only on classification tasks. All tasks are evaluated using a validation split, except for IMDB, where we take the mean score of five cross-validation folds. To make the validation faster to compute, we downsample the validation and train splits to 10000 examples. We downsample the datasets following their label distribution so that the truncated split has a label distribution nearly identical to the original one. We present the validation tasks and their document count in Table 4.2.

We use binary or micro-averaged accuracy as the scoring metric. Often, compare performance across several tasks. However, not all tasks are equally difficult, so averaging accuracies would lead us to favor models that performed well on easy tasks and undervalue models that performed well on more difficult tasks. Therefore, we normalize the accuracy by the highest score reached for the given task within the considered models, making the tasks equally difficult. We call this metric *normalized accuracy*. When more tasks are taken into account, we asses models based on the mean normalized accuracy. In visualizations, we mark mean normalized accuracy with a black triangle.

When validating a trained embedding model on a task, we finetune a head that transforms the embeddings into the output format required for the given task. We do not finetune the embedding model itself. Besides speeding up the validation, this gives us a more genuine picture of the embedding model's performance. Since all our validation tasks are classifications, the heads are just 2-layer neural network classifiers with a cross-entropy loss. We present the complete list of the classifier's hyperparameters and training parameters in Table 4.3.

Dataset	Documents		Classes	Class percentage	
	Train	Validation		Train	Validation
ARXIV [He et al., 2019]	†10 000	2 500	11	9.09±1.24%	9.09±1.01%
IMDB [Maas et al., 2011]	†10 000	-	2	50.00±0.00%	-
OC [Xuhui Zhou, 2020]	†10 000	†10 000	2	50.00±0.06%	50.00±0.15%
AAN [Xuhui Zhou, 2020]	†10 000	†10 000	2	50.00±1.50%	50.00±4.57%
S2ORC [Xuhui Zhou, 2020]	†10 000	†10 000	2	50.00±0.09%	50.00±0.32%
PAN [Xuhui Zhou, 2020]	†10 000	2 908	2	50.00±0.00%	50.00±0.00%

Table 4.2: Validation tasks we use to compare embedding models in this chapter. We truncated splits marked with † to speed up the evaluation process. We truncate a split by downsampling it following its label distribution. We also show the mean and standard deviation of class percentages to show all tasks have failry balanced class distributions.

Parameter	Value
Hidden features	50
Hidden dropout rate	0.5
Hidden activation	ReLU
Epochs	10
Batch size	32
Weight decay	0.1
Label smoothing	0.1
Learning rate	1e-4
Learning rate decay	Cosine
Maximum gradient norm	1.0
Optimizer	AdamW
Mixed-precision training	Yes

Table 4.3: Hyperparameters used for training classification heads during evaluation in this chapter.

Parameter	Value
Batch size	6
Weight decay	0.1
Learning rate	1e-4
Learning rate decay	Cosine
Maximum gradient norm	1.0
Optimizer	AdamW
Gradient accumulation steps	1
Warmup steps	10% of training steps
Gradient checkpointing	Yes
Mixed-precision training	Yes

Table 4.4: Training parameters' values we use every time we train a student model in this chapter.

4.3 Student model's configuration and baselines

As we explained in Section 3.3, we initialize our student model with Longformer [Beltagy et al., 2020]. We use Longformer's base version with about 126M parameters implemented by HuggingFace `transformers` library[1].

We pool the last layer's hidden states and compute their mean to obtain the input's embedding. We do not use global attention and employ sliding window attention, with the window sizes ω set to the default 512 tokens. In our preliminary experiments, we also tested using global attention to the `CLS` token and taking its hidden state from the last layer as the input's embedding. However, the mean-pooling approach proved to be superior. Additionally, with mean-pooling, we found global attention is not beneficial, so we avoided it.

We aim for fast convergence with a small memory footprint when training the student model. We, therefore, use a high learning rate, no gradient accumulation steps, mixed-precision training, and gradient checkpointing. We enumerate the complete list of student's training parameters in Table 4.4. We use these values for all student's training in this chapter.

4.3.1 Baselines

As mentioned, we aim to finetune our training method so the student model surpasses teachers and Longformer. To check how close we are to this goal throughout this chapter, we compare the student variants to three models: Longformer [Beltagy et al., 2020], SBERT [Reimers and Gurevych, 2019], and PV [Le and Mikolov, 2014]. We compare students to Longformer to judge how our training method improves document embeddings. As we mentioned above, we use a subset of Longformer's pre-training data. So, the student model cannot gain performance just due to the training data, as it has already seen them during pre-training. For context, we train the student models for only 3.8% iterations of Longformer's pre-training with an eight times smaller batch size.

We compare the students to the two teachers to see how much performance our training method ignores or takes advantage of. If our student model performs worse than a particular teacher, we need to improve how we distill the teacher's

[1] `https://huggingface.co/allenai/longformer-base-4096`

embeddings into the student's embeddings. As the student has architectural advantages compared to both teachers, such as longer maximum context, we hypothesize it can match and surpass both teachers' performance. We discuss the configuration of the structural teacher in the following section. Paragraph Vector's training is a part of our training method, so we discuss further details in Section 4.5.1, where we experiment with PV's training hyperparameters.

4.4 Structural loss

We start our experiments with the structural loss $\mathcal{L}_S$. The structural loss compares the student's and the structural teacher's embeddings. Its goal is to encourage distillation of the quality of the structural teacher's embeddings into the student's embeddings. We focus on the structural loss first since, in our preliminary experiments, we observed that the structural quality is more significant to the performance of the student model than the contextual quality. We arrive at the same conclusions later in this chapter in Section 4.5.2. Therefore, we prioritize first finding the best-performing hyperparameters of the structural loss and adapting the contextual loss to it afterward.

As we mention in Section 3.2, we use SBERT [Reimers and Gurevych, 2019] as our structural teacher. We use SBERT's version initialized with MPNet [Song et al., 2020] since it is a relatively small model with above-average performance[2]. We use SBERT's implementation from the HuggingFace `transformers` library[3] with a mean pooling layer above the last layer's hidden states. We do not perform any finetuning and use the pre-trained weights only.

As explained in Section 3.1.2, we choose structural loss to be more restrictive, forcing an exact similarity between the two embeddings. Therefore, we test two different exact losses as structural losses: Mean Squared Error (*MSE*) and cosine distance. We try MSE because it forces equality, the most restrictive similarity measure. The motivation for using cosine comes from the embeddings' use cases, where cosine distance is a popular similarity measure. More importantly, it is also used by SBERT's authors, suggesting that for SBERT's embeddings, it is the best-performing similarity measure.

We train the student model with each loss on the first 15k documents of VAL-500K with the hyperparameters given in Section 4.3. As we show in Figure 4.2, with cosine distance, the student model surpasses both baselines. The fact that the student performs better than the teacher indicates that Longformer's longer context and supervision from the structural teacher can boost the student's performance even above the level of the teacher. Moreover, SBERT's scores are significantly higher than Longformer's, showing that unless Longformer's embeddings are finetuned, its longer context length or pre-training is of no benefit.

These encouraging results show that using only the structural teacher can be a valid method of improving a model's embeddings. However, we hope to enhance the student's performance even further. In subsequent experiments, we use cosine as the structural loss. For brevity, we label the student model trained with only the structural loss as only-structural;cosine.

[2] `https://sbert.net/docs/pretrained_models.html`

[3] `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`

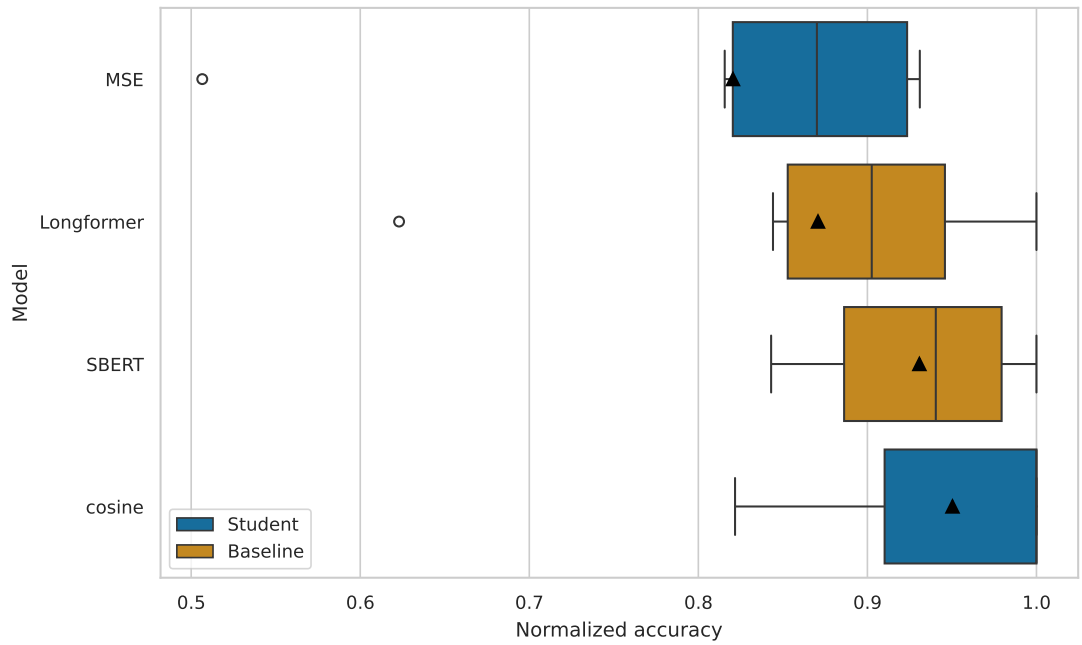

Figure 4.2: Performance of student models trained with only the structural teacher.

4.4.1 Composite structural losses

Besides the cosine and the MSE, we also explore losses that combine a positive and a negative component, such as contrastive loss. We call these losses *composite* to differentiate them from *simple* losses, such as MSE or cosine distance. Composite losses compare the student's embedding to a teacher's embedding of the same input, which we call *positive*, and to the teacher's embedding of different inputs, which we call *negatives*. They reward the student model for close proximity to positives or large distances to the negatives. Therefore, structural composite losses optimize two aspects: they decrease the distance to SBERT's embeddings while increasing the margin between the student's embeddings of different inputs.

We explore two types of composite losses: max-margin and contrastive. To formulate these losses, we label the student's embedding as y, the corresponding teacher's embedding as y_{pos}, the set of negatives as Y_{neg}, the given similarity measure as sim, and a weighting parameter as γ. We define the max-margin loss in Equation 4.1 and the contrastive loss in Equation 4.2.

$$\mathcal{L}_{\text{max-margin}}(y, y_{\text{pos}}, Y_{\text{neg}}) = \text{sim}(y, y_{\text{pos}}) - \gamma \frac{1}{|Y_{\text{neg}}|} \sum_{y_{\text{neg}} \in Y_{\text{neg}}} \text{sim}(y, y_{\text{neg}}) \tag{4.1}$$

$$\mathcal{L}_{\text{contrastive}}(y, y_{\text{pos}}, Y_{\text{neg}}) = -\log \frac{\exp(\cos(y, y_{\text{pos}}))}{\exp(\cos(y, y_{\text{pos}}) + \sum_{y_{\text{neg}} \in Y_{\text{neg}}} \cos(y, y_{\text{neg}}))} \tag{4.2}$$

For max-margin loss, we try MSE and cosine distance as sim and simultaneously try several weightings γ. We compare models trained with the composite losses with those trained with the simple losses in Figure 4.3. Composite losses

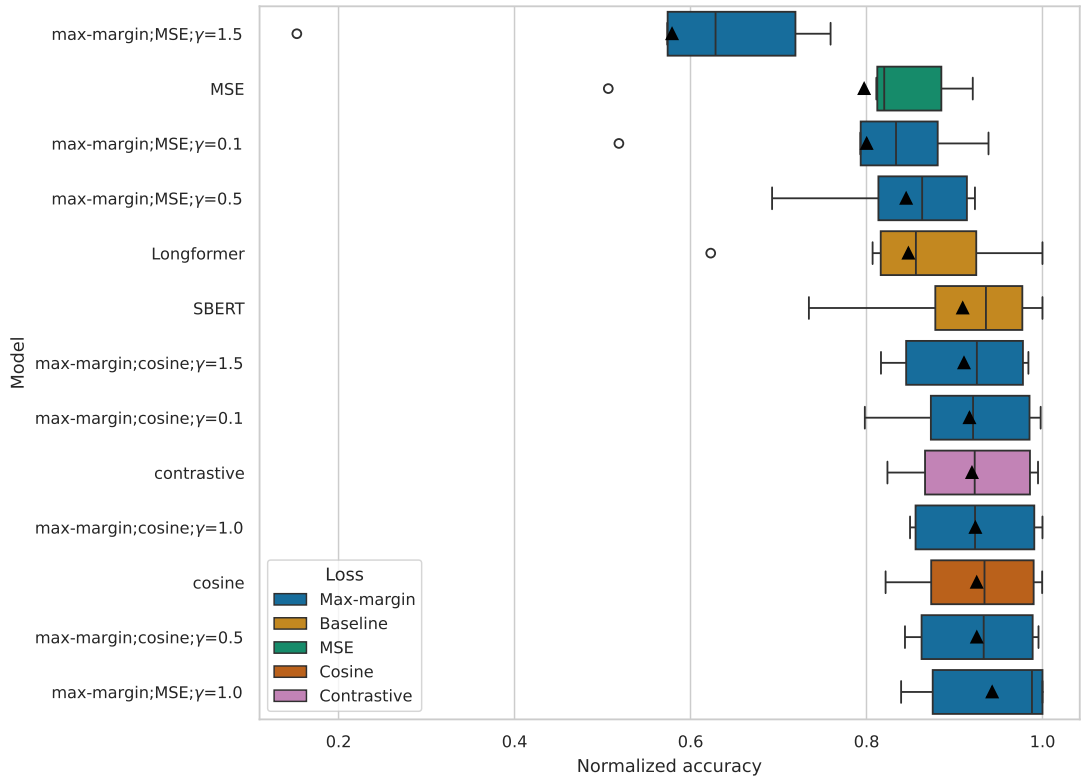

Figure 4.3: Performance of student models trained with composite and simple structural losses.

using MSE seem to benefit from the negative loss component, as 3 out of 4 outperform the simple MSE loss. However, despite the benefits, only one variant surpasses both baselines. With cosine, it seems that the negative loss component hurts the performance since only one version outperforms the simple cosine loss and does so by only 10^{-4}. We carry out an analysis, to explain the differences between MSE and cosine composite losses and to show how the negatives contribute to the model's performance.

Analysis of composite losses

The impact of composite losses is different for MSE and cosine. Also, it is not clear how the negatives contribute to the student's performance. To explain these results, we compare the distances to positives and negatives for several chosen models in Figure 4.4. The plotted distances nicely mirror the students' performances. However, the effect of the negative loss component is minimal for most student models. Except for max-margin;MSE with γ set to 1 or 1.5, there is no dramatic shift in the distances' distributions. In terms of squared L2 distance, max-margin;MSE;γ=1.0 widens the gap between positives and negatives, yet it also increases the distances to the positives. However, this effect is much less pronounced for cosine distance. So, the model increases the embeddings' norm to create a large gap between positives and negatives in terms of squared L2 distance, while decreasing the cosine distance to positives. In other words, despite computing L2 distances, max-margin MSE loss with $\gamma = 1.0$ optimizes cosine distance. Note that with $\gamma = 1.5$ the negative loss component has damaging

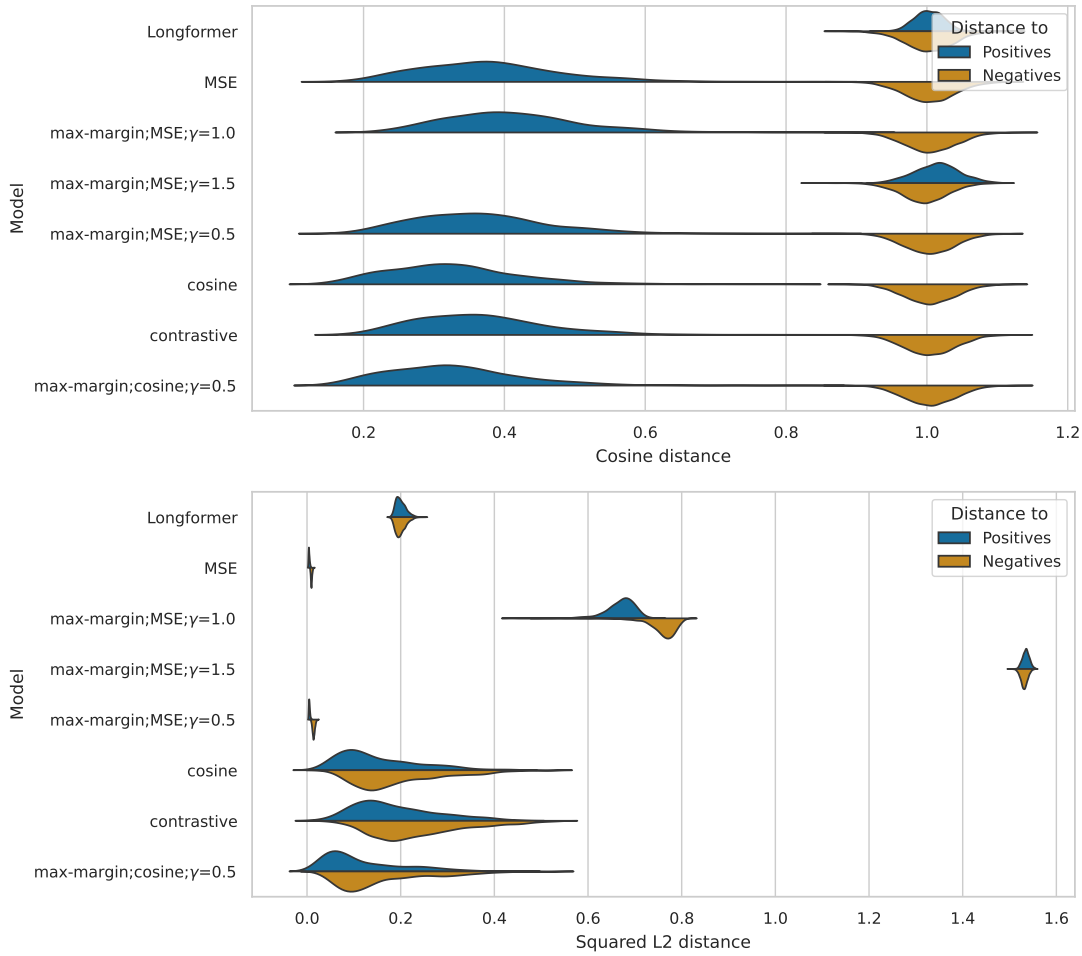

Figure 4.4: Distribution of distances between the model's and the structural teacher's embeddings. A distance to the teacher's embedding of the same document is labeled as *positive*, whereas distances to the teacher's embedding of another document are labeled as *negative*. We generated the distances from the first 1000 documents of VAL-500K's validation split.

effects in terms of both cosine and squared L2 distances.

To summarize, the effect of the negatives being included in the loss may be dual. With the right configuration, the composite losses may enforce larger separation between student's embeddings, while simultaneously decreasing their distance to the structural teacher's embeddings. So, besides only-structural;cosine we also continue experimenting with max-margin;MSE;γ=1.0, which we label in further sections as only-structural;mm-MSE.

4.5 Structural and contextual loss

This section explores contextual losses that complement the best-performing structural losses from previous section. Since there are several hyperparameters to explore, we split this section into parts. Each part focuses on a different aspect of the contextual loss or the weighting of the contextual and structural loss. At the end of each part, we select a few best-performing variants, which

serve as a starting point in the subsequent section. First, we test different training hyperparameters of the contextual teacher in Section 4.5.1. Then, we experiment with the contextual loss's configuration in Section 4.5.2. To illustrate the importance of the structural teacher, we show the performance we reach when we use only the contextual loss. More importantly, we test the contextual losses with both structural losses and select the best performing configuration for each one. Finally, we explore different ways to weigh the contextual and structural loss for both combinations in Section 4.5.3.

4.5.1 Optimizing Paragraph Vector's training

We choose Paragraph Vector [Le and Mikolov, 2014] as our contextual teacher, as we elaborate on in Section 3.2.2. Since there is no concept of a pre-trained PV, as in the case of transformers, we train PV from scratch. We use PV's implementation from the Gensim library[4] and explore some of the hyperparameters that govern the training of PV. We focus on four hyperparameters that we consider important and adopt the recommendation of the library or related literature. We enumerate the adopted and the grid-searched hyperparameters in Table 4.5. To explain the meaning of all hyperparameters, we provide the following summary:

- `dm` – PV architecture; true for Distributed Memory (DM), false for Distributed Bag of Words (DBOW)
- `vector_size` – dimensionality of the generated embedding
- `min_count` – words with document frequency below this limit will be ignored
- `text_pre_process` – applied word processing done before the model's training; for stemming, we use PorterStemmer implemented by the `nltk` library[5]
- `negative` – number of noise words used for negative sampling during training
- `window` – the maximum distance between known and predicated word
- `sample` – percentile threshold configuring which words will be downsampled; 0 for no downsampling
- `dbow_words` – whether to train word embeddings using Word2Vec's [Mikolov et al., 2013] Skip-gram architecture together with document embeddings; only applicable to DBOW, as DM learns word embeddings by default
- `epochs` – a number of iterations done over the corpus during training

As recommended by the authors of PV [Le and Mikolov, 2014], we experiment with both architectures. For each architecture, we try different values of `vector_size`, `min_count`, and `text_pre_process`, which all control the model's regularization. Settings such as higher dimensional embedding, small minimum count, and no text pre-processing regularize the model the least. They give the

[4] `https://radimrehurek.com/gensim`

[5] `https://www.nltk.org/api/nltk.stem.porter.html`

Hyperparameter	Value(s)	Recommended by
`dm`	true, false	-
`vector_size`	100, 768, 1024	-
`min_count`	2, 10% of training corpus	-
`text_pre_process`	stem, lowercase, none	-
`window`	5	default
`negative`	5	default, Lau and Baldwin [2016]
`sample`	0	default
`dbow_words`	true	Lau and Baldwin [2016]
`epochs`	10	default, Dai et al. [2015]

Table 4.5: Used hyperparameters for training Paragraph Vector. We grid-searched four hyperparameters: PV architecture, vector size, minimum word count, and pre-processing of words. For the rest of the hyperparameters, we adopted either the default values or recommended by the mentioned literature.

model the most information on its inputs while providing it with large embedding through which it can express precisely. On the other hand, using lower dimensional embedding, large minimum count, and stemming the document's words forces the model to be more general and less precise. The model has less detailed information on its input and must squeeze all of it into a small vector. We do not see any value in trying dimensions of embeddings higher than 1024 since, in later experiments, we must distill the contextual embedding to a 768-dimensional embedding of our student model. Intuitively, the larger the contextual embedding will be, the smaller the fraction of information the student model will be able to digest. Also, there is no value in considering `min_count` to be lower than two since we would only add words unique to a single document. Embeddings of such words would be poorly trained and not add meaningful information to the document's embedding. The last hyperparameter that is worth mentioning is `dbow_words`. DBOW, on its own, does not train word embeddings, which are, by default, randomly generated. Setting `dbow_words` to true causes DBOW also to train word embeddings using Word2Vec's Skip-gram model [Mikolov et al., 2013] in each epoch. Lau and Baldwin [2016] showed that random word embeddings significantly hurt the model. Consequently, when training DBOW, we also train word embeddings despite the slower training, which it inevitably causes.

We train all variants on the whole VAL-500K corpus. We follow the recommendations of Le and Mikolov [2014] and also evaluate the combination of both architectures. However, we only select the best three models from each architecture and evaluate all nine combinations. We call these models *compound* Paragraph Vectors. In total, there are 45 models whose performance on validation tasks we report in Figure 4.5. The single models favor large embedding dimensions and low minimum count. Additionally, on average, stemming or lowercasing leads to higher scores than not pre-processing the words. DBOWs vary more in performance, occupying the best and the worst positions, whereas DMs are more consistent. We achieve a slight improvement by concatenating a DM and a DBOW model, but considering the resulting model has an embedding twice as large, the improvement is not surprising. Interestingly, all compound models perform very similarly, suggesting that slight imperfections of one model can be compensated by another model of a different architecture.

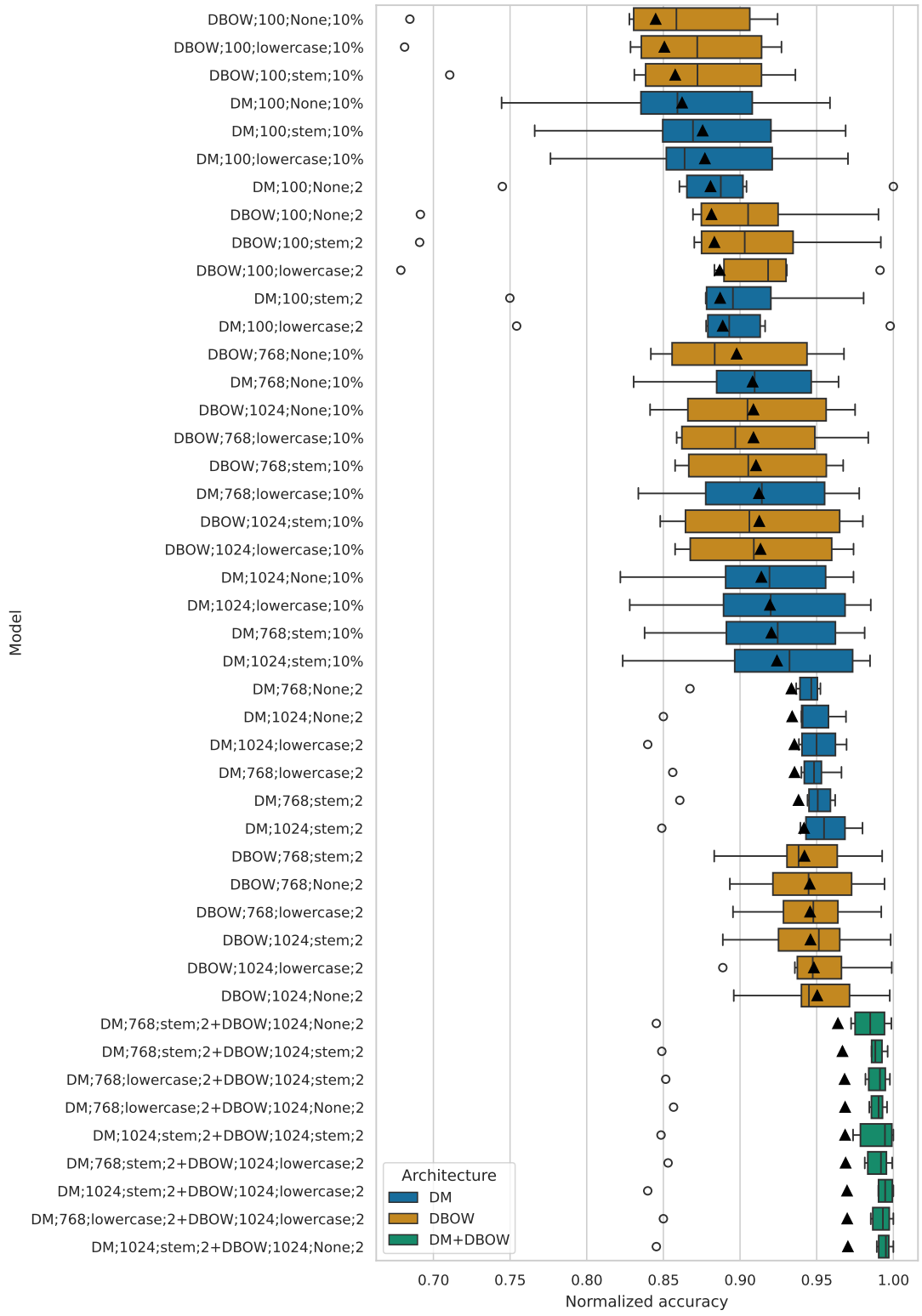

Figure 4.5: Performance of all Paragraph Vector variants on validation tasks. We identify a model by its architecture, embedding dimension, text pre-processing, and minimum count. Compound models are identified as a concatenation of such identifiers separated by +.

In our preliminary experiments, we saw that the dimension of contextual teacher embedding plays a significant role in teacher-student training. So, we select three paragraph vectors with varying vector sizes. We pick the best model with small vector size (DM;100;lowercase;2), the best single model (DBOW;1024;None;2) and the best model composed of both architectures (DM;1024;stem;2+DBOW;1024;None;2). For brevity we label these models as DM;100d, DBOW;1024d and PV;2048d respectively.

4.5.2 Contextual loss

Contextual loss $\mathcal{L}_C$ compares the student's and the contextual teacher's embeddings and encourages distillation of the quality of the teacher's embedding into the student's embedding. As we discuss in Section 3.1.2, we choose $\mathcal{L}_C$ to be less strict and give the student model more freedom in encoding information into the document embedding. Consequently, we do not consider losses such as MSE or cosine since they enforce either an exact vector or a direction in the embedding space. Instead, we use a variant of *Canonical Correlation Analysis* [Hotelling, 1992] (*CCA*). In its base form, CCA computes a correlation of two linearly projected sets of vectors, where the projections are optimized to maximize the correlation. We define CCA in Equation 1.

Definition 1 (Canonical Correlation Analysis)**.** *For two matrices $X_1 \in \mathbb{R}^{n_1 \times m_1}$ and $X_2 \in \mathbb{R}^{n_2 \times m_1}$, Canonical Correlation Analysis for k dimensions finds $P \in \mathbb{R}^{m_1 \times k}$ and $Q \in \mathbb{R}^{m_2 \times k}$ that maximize*

$$\begin{aligned} \mathrm{CCA}(X_1, X_2) &= \sum_{i=1}^{k} \mathrm{corr}(X_1 P_{*i}, X_2 Q_{*i}) \\ s.t. \quad P^T X_1^T X_1 P &= I_k = Q^T X_2^T X_2 Q \end{aligned} \tag{4.3}$$

CCA gives the student the freedom to change its embeddings as long as their linear projections correlate more with the linear projections of the contextual teacher's embeddings. However, linear projection may still leave too little leeway for the student model to simultaneously mimic the structural teacher while increasing the correlation with the projected contextual teacher's embeddings. Ideally, we would like to regulate the strength of the projections. Such adjustment is possible with *Deep CCA* (*DCCA*) [Andrew et al., 2013]. DCCA projects the input vectors with two neural networks and feeds the projections to the vanilla CCA. The two networks are trained jointly with the embedding model based on the computed CCA, which is used as a loss. The advantage of DCCA is that we can adjust the strength of the projections and thereby regulate the pressure the contextual loss inflicts on the student model. The larger the neural network is, the more it can transform the embeddings and the less the student model needs to adjust its embedding.

As we can see in Equation 1, CCA is computed from the entire dataset of input vectors. And so DCCA is trained using a full-batch optimization [Andrew et al., 2013] or a mini-batch optimization with large batch sizes [Wang et al., 2015]. However, both methods need large amounts of GPU memory and are suitable only with smaller models. For this reason, we avoid CCA and use SoftCCA [Chang

et al., 2018] instead. SoftCCA reformulates CCA such that it is usable even in the case of mini-batch optimization with small batches. To explain how SoftCCA is related to CCA, we reformulate the solution to CCA using a Forbenious matrix norm in Equations 4.4-4.8.

$$
\begin{aligned}
P^*, Q^* &= \underset{P,Q}{\operatorname{argmin}} \, ||X_1 P - X_2 Q||_F^2 && (4.4)\\
&= \underset{P,Q}{\operatorname{argmin}} \, \operatorname{trace}\Big((X_1 P - X_2 Q)^T (X_1 P - X_2 Q)\Big) && (4.5)\\
&= \underset{P,Q}{\operatorname{argmin}} \, -2 \operatorname{trace}(P^T X_1^T X_2 Q) && (4.6)\\
&= \underset{P,Q}{\operatorname{argmax}} \, \operatorname{trace}(P^T X_1^T X_2 Q) && (4.7)\\
&= \underset{P,Q}{\operatorname{argmax}} \sum_{i=1}^{k} \operatorname{corr}(X_1 P_{*i}, X_2 Q_{*i}) && (4.8)
\end{aligned}
$$

Thus, by minimizing CCA, we effectively minimize the difference between two projections that have uncorrelated features. SoftCCA enforces the same behavior with two separate losses:

- L2 loss, which minimizes the difference between projected set of vectors Z_1 and Z_2:

$$\mathcal{L}_{\text{L2}}(Z_1, Z_2) = ||Z_1 - Z_2||_F^2 = \text{MSE}(Z_1, Z_2) \tag{4.9}$$

- *Soft Decorrelation Loss* (*SDL*), which forces a projected set of vectors Z to have decorrelated features:

$$\mathcal{L}_{\text{SDL}}(Z^t) = \sum_{i \neq j} \left| \frac{(\Phi_Z^t)_{ij}}{\hat{\beta}^t} \right| \tag{4.10}$$

  where

$$
\begin{aligned}
\Phi_Z^t &= \beta \Phi_Z^{t-1} + \Sigma_{Z^t} && (4.11)\\
\Phi_Z^0 &= \mathbf{0}_d && (4.12)\\
\hat{\beta}^t &= \beta \hat{\beta}^{t-1} + 1 && (4.13)\\
\hat{\beta}^0 &= 0 && (4.14)
\end{aligned}
$$

Where the symbols above have the following meaning:

- $Z, Z_1, Z_2 \in \mathbb{R}^{b \times d}$ are mini-batches of d-dimensional vectors
- Σ_Z is a covariance matrix of a mini-batch of vectors Z
- $\mathbf{0}_d$ is a $d \times d$ zero matrix
- β is a hyperparameter
- $\Phi_Z^t, Z^t, \hat{\beta}^t$ is $\Phi_Z, Z, \hat{\beta}$ at iteration t

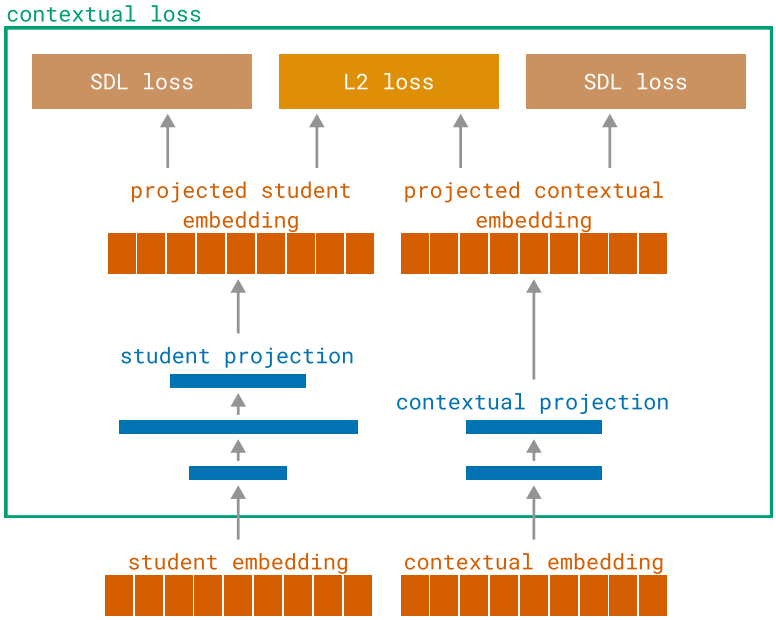

Figure 4.6: Architecture of contextual loss.

The L2 loss forces the projected vectors to be equal, while the Soft Decorrelation Loss forces them to have uncorrelated features. SoftCCA is able to approximate CCA, by keeping a running mean of the projected vectors' covariance matrices. Similarly to Batch Normalization [Ioffe and Szegedy, 2015], SDL updates the running mean during training but avoids any updates during inference. With the above losses and the weighting hyperparameter δ, we define SoftCCA in Equation 4.15.

$$\mathcal{L}_{\text{SoftCCA}}(Z_1, Z_2) = \mathcal{L}_{\text{L2}}(Z_1, Z_2) + \delta(\mathcal{L}_{\text{SDL}}(Z_1) + \mathcal{L}_{\text{SDL}}(Z_2)) \tag{4.15}$$

We use SoftCCA loss as a replacement for CCA in DCCA. To be explicit, we project the student's and the contextual teacher's embedding with two separate feed-forward neural networks. Then, we apply SoftCCA loss, which provides a training signal for the embedding model and the neural networks projecting the embeddings. For brevity, we call the neural networks projecting an embedding *student* and *contextual projections*, depending on which embedding they project. And so, we can finally express $\mathcal{L}_C$ from Equation 3.1 more concretely. For a student projection f_S and a contextual projection f_C, we formulate our contextual loss in Equation 4.16. We also illustrate the architecture of contextual loss graphically in Figure 4.6.

$$\mathcal{L}_C(y_S, y_{\mathcal{T}_C}) = \mathcal{L}_{\text{SoftCCA}}(f_S(y_S), f_C(y_{\mathcal{T}_C})) \tag{4.16}$$

In our preliminary experiments, we found that the value of δ from Equation 4.15 has little effect on the final student's performance. So, we set it so that the ranges of $\mathcal{L}_{\text{L2}}$ and $\mathcal{L}_{\text{SDL}}$ are roughly equal. On the other hand, choosing the right value of β proved to be critical. Based on how the CCA of the projected embeddings of the validation split progressed throughout the training, we found the optimal value to be 0.95, which puts a relatively large emphasis on the accumulated mean compared to lower values of β.

In the rest of this section, we experiment with the strength of the projections. In the following section, we build the basic intuition behind training with the contextual loss, while finding the ideal projections without any structural loss. In the subsequent sections, we study how different structural losses influence the projections. First, we experiment with cosine and then with max-margin MSE loss. In all cases, we test the projections for each of the three contextual teachers: DM;100d, DBOW;1024d and PV;2048d. Finally, we select the best contextual loss with the best contextual teacher for both cosine and max-margin MSE structural loss.

Contextual projection with contextual loss only

With only the contextual loss, the student's only goal is to mimic the contextual teacher. This presents an elementary setting in which we can study the behavior of the projections and the contextual loss as a whole without any influence from the structural loss.

Our preliminary experiments show that the projections' over-parameterization hurts the model's performance. Even though large projections result in smaller SoftCCA loss, they tend to harm the CCA computed on the student's and contextual teacher's embeddings. Strong projections compensate for the student's flaws, lessening the pressure on the student model as it does not need to adjust its embedding much. Consequently, the student model learns very little compared to the projections. Similarly, strong contextual projection takes away pressure from the student projection and vice-versa. In this regard, it is essential to keep the contextual projection small. This puts more pressure on the student's side, where the gradients can propagate to the embedding model.

As we mentioned before, we feed the projected outputs to SoftCCA loss $\mathcal{L}_{\text{SoftCCA}}$. As SoftCCA requires both inputs to be of the same dimension, both projections must end with an equally sized layer. We always use the dimension of the larger embedding as the final projection dimension. We do so to preserve all the embeddings' information through the projection and force the student model to distill all contextual embedding's dimensions, not just their subset. Also, there is no point in projecting the embeddings to even more dimensions than the embeddings have. Due to the Pigeonhole principle, some features of the final projections would have to depend on the same embedding's features and, therefore, would correlate with each other. Such correlations would create unnecessary conflict with the SDL loss. This phenomenon would also occur for projections with an hourglass shape, where there is one bottle-neck layer with significantly fewer dimensions than the layers after or before it. And indeed, during preliminary testing, we saw these projections always perform poorly.

We build the projections as a sequence of blocks, where each block is composed of a fully connected layer and an optional Rectified Linear Unit ($ReLU$). In preliminary experiments, we also tried adding Dropout, Batch, or Layer Normalization layers at different places in a block. However, in all cases, they had either negligible or negative effects on the performance of the final model. We label each block with the dimension of the fully connected layer and with the activation's name in brackets if used. We identify a projection by block's labels delimited by an "x". So, 768(ReLU)x1024 are two feed-forward layers with 768 and 1024 features connected via ReLU. To label projections without any layers, we use

	Contextual teacher's embedding dimension		
Projection	100	1024	2048
Student	768(ReLU)×1024(ReLU)×768	768(ReLU)×1024	1024(ReLU)×2048
	768	1024	2048
Contextual	100(ReLU)×768	768×1024	1024×2048
	768	1024	2048
		-	-

Table 4.6: All tested variants of projections with only contextual loss. We do a grid search of the given variants for each contextual teacher. This results in 16 combinations overall.

a dash. We present all the projections' variations we tested in Table 4.6. Considering the conditions described in the previous paragraph, we choose a strong and a weak projection for both the student and contextual side. We are careful not to over-parametrize either projection and lean toward stronger student projection.

We train the student models on the first 15k documents of VAL-500K and compare the models' performance to all the relevant teachers, Longformer and only-structural;cosine in Figure 4.7. We identify a student model with the contextual teacher's dimension, the student projection prefixed by S: and the contextual projection prefixed by C:.

The results correspond to those we witnessed in our preliminary experiments and showcase some of the mentioned projections' behaviors. The better half of the student models differs from the rest by having a minimal contextual projection. Moreover, for a given contextual teacher and a projection, the student model with a larger student projection outperforms the student with a smaller one in all cases but one.

Half of the tested projections improve the score of Longformer. The better projections demonstrate that we can distill useful information from the contextual teacher to the student, while the worse projections highlight how important the projections are. However, as our contextual loss does not enforce an exact similarity of the student's and the contextual teacher's embedding, most students do not outperform their respective contextual teachers. Interestingly, students trained with DM;100d surpass students trained with better performing DBOW;1024d. We can observe the same performance differences for DBOW;1024d and PV;2048d, even if we compare projections that scale with the contextual embedding, such as 1024d;S:1024;C:- and 2048d;S:2048;C:- or 1024d;S:1024;C:1024 and 2048d;S:2048;C:2048. Consequently, distilling information from an embedding with fewer features seems easier than from a larger one. And so, even though PV with 2048 dimensions beats SBERT, the students trained with it fail to capitalize on this advantage and perform worse than the model trained with SBERT. Therefore, we conclude that according to our results, the structural teacher is more important to the student's performance than the contextual teacher and justifies why we search for the ideal contextual loss to the best performing structural loss rather than the other way around.

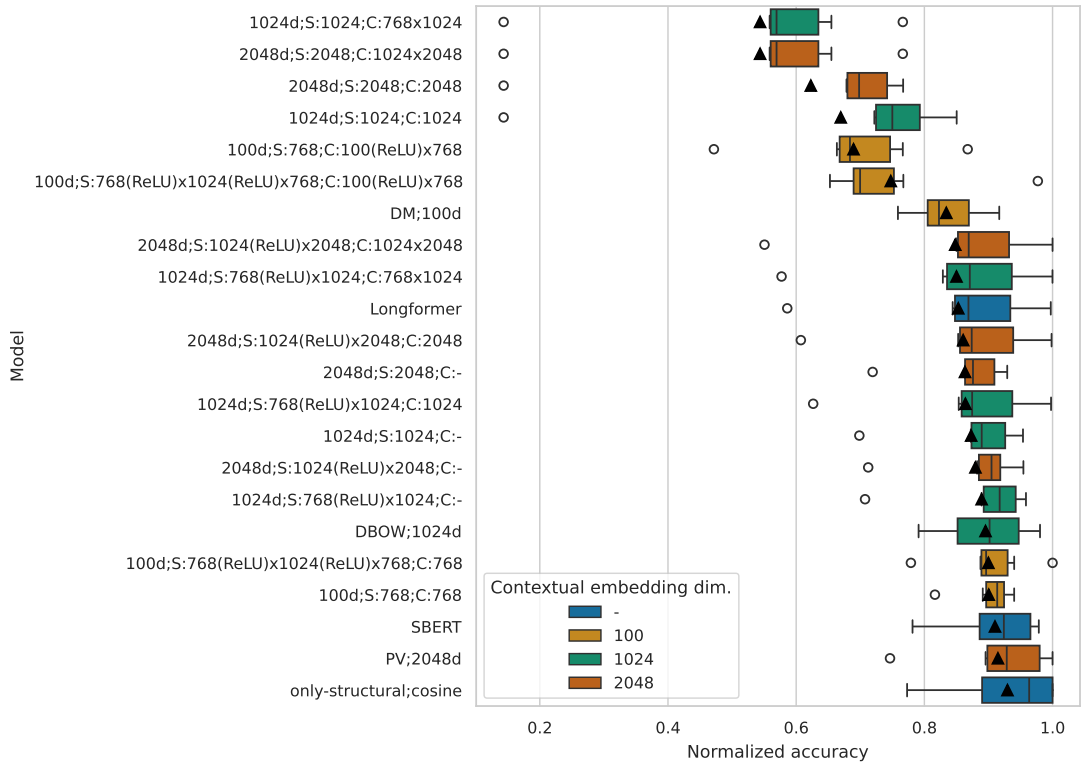

Figure 4.7: Performances of student models trained with different projections and without structural loss. We compare the students to all the relevant teachers, Longformer, and only-structural;cosine.

Contextual projection with cosine structural loss

In this section, we search for the best-performing projections while simultaneously using cosine structural loss. The training is a bit more complex than in the previous section as here the student should distill two qualities, each from a different teacher. So, even though the student and contextual projections still behave equally, they may cause different outcomes.

We list all the tested projections in Table 4.7. We add stronger projections while discarding some of the less successful projections from the previous section. We again train the student models on the first 15k documents of VAL-500K and present the performance of the trained student models in Figure 4.8. The projection variants differ less than in the case of the student models trained without structural loss. The cosine structural loss boosts the models' performance and lessens the negative impact of a poor projection. As a consequence, almost all of the student models surpass all baselines. The best-performing projections are much larger compared to the best projections trained without any structural loss. This seems logical, as the added structural loss puts more pressure on the student, and so the contextual loss needs to give the model more freedom in order to avoid conflict between the losses, which would slow down the training. With the larger projections, we were able to surpass only-structural;cosine. This shows that, with the right projections, the student model benefits from both losses being used during training. Even if the performance gain is not huge, we conclude that the contextual and structural embeddings may complement each other in the

	Contextual teacher's embedding dimension
Projection	100
Student	768(ReLU)×1024(ReLU)×768
Contextual	100(ReLU)×768
	768

(a) 100-dimensional contextual teacher

	Contextual teacher's embedding dimension	
Projection	1024	2048
Student	768(ReLU)×1024	1024(ReLU)×2048
Contextual	1024	2048
	-	-
Student	768(ReLU)×4096(ReLU)×1024	768(ReLU)×4096(ReLU)×2048
Contextual	768(ReLU)×1024	2048(ReLU)×2048
	1024	2048
	-	-

(b) 1024 and 2048-dimensional contextual teachers

Table 4.7: All tested variants of projections with contextual loss and cosine structural loss. For a given contextual teacher, we delimit each group of projections by a horizontal line. We grid search all variants within each group. This results in 12 combinations of projections.

right setting.

Contextual projection with max-margin MSE structural loss

Finally we find the optimal projections with the max-margin MSE structural loss. We present all the tested projection variants in Table 4.8. We include successful projections from the previous section and add stronger contextual projections as they perform surprisingly well in this context. Same as before, we train all student models on the first 15k documents from VAL-500K and compare their performance to all relevant teachers, Longformer, and only-structural;mm-MSE. We present the model's performances in Figure 4.9. As with the cosine structural loss, max-margin MSE loss boosts the students' performances. Consequently, the student's performances are not as dependent on the projections as those of students trained without structural loss. Contrary to what we witness in previous experiments, stronger contextual projections perform very well overall. This shows that max-margin MSE loss more pressure on the student's embedding than the cosine structural loss. Despite testing even more projections than in the case of cosine structural loss, we fail to find projections which would outperform only-structural;mm-MSE. Nonetheless, we continue experimenting with the best projections we find.

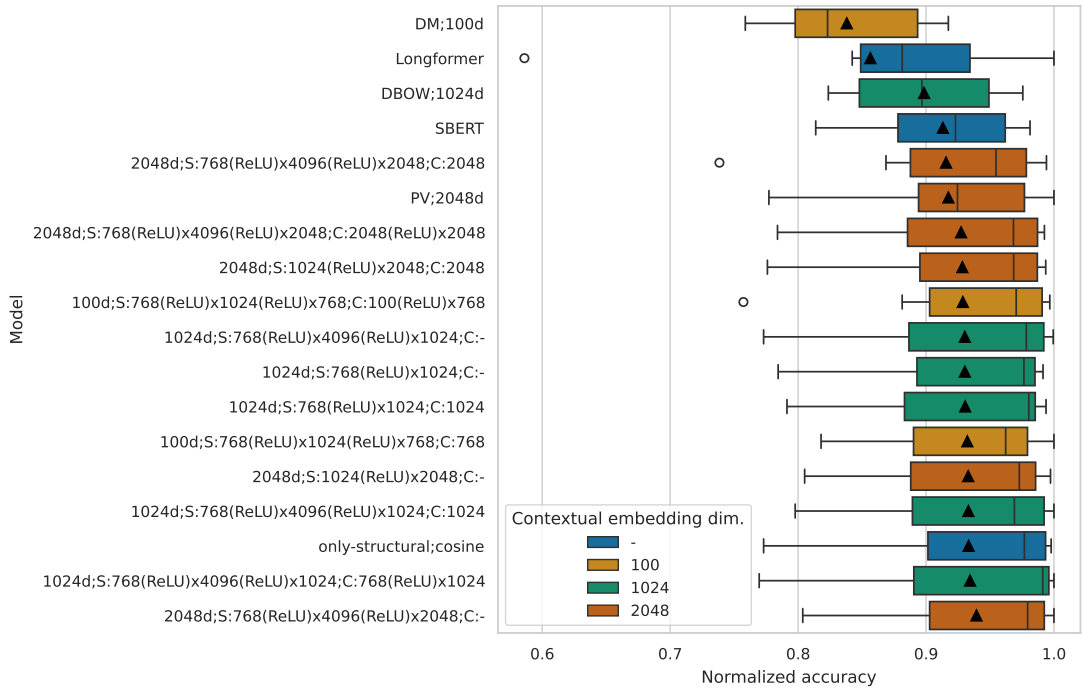

Figure 4.8: Performance of student models trained with contextual and cosine structural loss on validation tasks. We compare the student models to all relevant teachers, Longformer and only-structural;cosine.

	Contextual teacher's embedding dimension
Projection	100
Student	768(ReLU)x1024(ReLU)x768
	768
Contextual	100(ReLU)x768
	768

(a) 100-dimensional contextual teacher

	Contextual teacher's embedding dimension	
Projection	1024	2048
Student	768(ReLU)x1024	1024(ReLU)x2048
Contextual	768x1024	1024x2048
	1024	2048
	-	-
Student	768(ReLU)x4096(ReLU)x1024	768(ReLU)x4096(ReLU)x2048
Contextual	1024(ReLU)x1024	2048(ReLU)x2048
	768(ReLU)x1024	-

(b) 1024 and 2048-dimensional contextual teachers

Table 4.8: All variants of projections tested with max-margin MSE structural loss. For a given contextual teacher, we delimit each group of projections by a horizontal line. We grid-search all variants within a group. This results in 14 combinations in total.

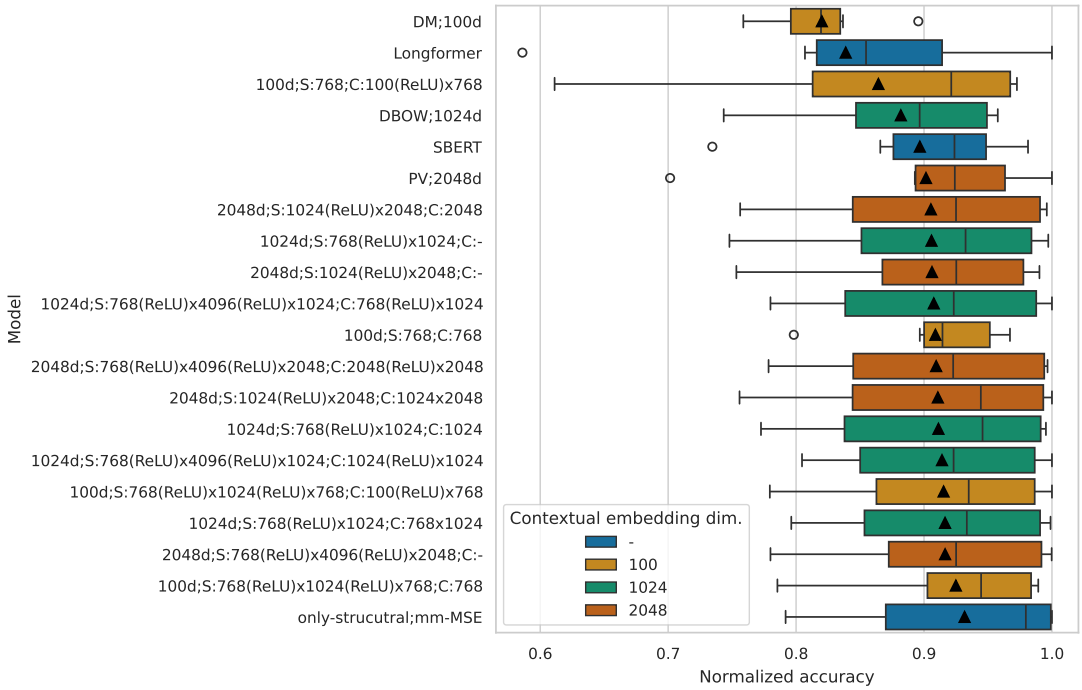

Figure 4.9: Performance of student models trained with contextual and max-margin MSE structural loss on validation tasks. We compare the student models to all relevant teachers, Longformer and only-structural;mm-MSE.

4.5.3 Weighting of structural and contextual loss

The final loss is a weighted sum of the contextual and the structural loss. In this section, we explore two weighting mechanisms. We combine a static weighting of each loss with dynamic masking of the structural loss based on inputs' length. As the structural teacher has limited context length, its embedding only reflects the information in the first 384 tokens. We use dynamic masking to train the student only on those inputs, which the structural teacher encodes whole. Therefore, in theory, the structural loss should be more reliable. To summarize, we grid-search two parameters: `max_structural_len` and λ. `max_structural_len` determines which inputs' structural loss we mask out. λ is the static weight used for unmasked inputs to balance the importance of the structural and contextual loss. For clarity, we include a Python-like pseudocode of the weighting algorithm in Listing 4.1. Note that we mask the inputs of the structural loss rather than its results. So, we effectively select only some inputs on which the loss will be computed. This is especially important for the max-margin loss, where the number of negatives effectively decreases.

In previous experiments, we weight the losses less intrusively. We do not mask out structural loss and sum the two losses. The advantage of this approach is that it does not reduce any gradients. However, we lose control over the mix of the two losses. Even if the losses are not weighted, we see this as another variant of obtaining the final loss and label it as no-weighting. We label all other weighting variants with the used `max_structural_len` and λ separated by a semicolon.

We consider two structural losses: cosine and max-margin MSE. For each structural loss, we take the best-performing contextual loss from the previous

```
length_mask = torch.ones(batch_size)
if max_structural_len is not None:
  length_mask = lengths <= max_structural_len
lams = torch.zeros(batch_size).fill_(λ)
lams *= length_mask

# For each loss we expect shape (batch_size,)
structural_loss = structural_loss_fn(..., mask=length_mask)
contextual_loss = ...

loss = structural_loss * lams + contextual_loss * (1 - lams)
loss = torch.mean(loss)
```

Listing 4.1: Python-like pseudocode of weighting algorithm.

max_structural_len	λ
384	0.95
None	0.8
	0.5
	0.2

Table 4.9: Tested weighting hyperparameters' values. We experiment with several static weightings λ with or without dynamic masking of structural losses for inputs longer than 385 tokens.

sections and try all combinations of weighting hyperparameters' values we list in Table 4.9. We train a student model with each weighting variant on the first 15k documents from VAL-500K.

Weighting a contextual and the cosine structural loss

We search for the best weighting configuration for the cosine structural loss, PV;2048d contextual teacher and S:768(ReLU)x4096(ReLU)x2048;C:- projections, which is the most promising combination. As we mention above, we label this combination without any weighting as no-weighting. We present all the models' performances in Figure 4.10. All the weighting variants surpass all baselines. Interestingly, even if the weighting is set significantly toward one side, such as None;λ=0.95 or 384;λ=0.2, the student model can surpass the other teacher. Therefore, the student can use the information provided by either teacher to surpass the other one. More importantly, the best weighting variants that beat only-structural;cosine are more cautious with the structural loss. They either mask it for longer documents or give it a smaller weight. Consequently, forcing the student model to distill embedding that captures only a partial part of its input confuses it, thereby hurting its performance.

We highlight the difference in performance between None;λ=0.5 and no-weighting. These models' losses are the same, except that None;λ=0.5 effectively halves all loss gradients. This results in a noticeable drop in performance. However, even with the gradients being halved, 384;λ=0.5 beats no-weighting variant. This further emphasizes the importance of masking out structural loss for long inputs.

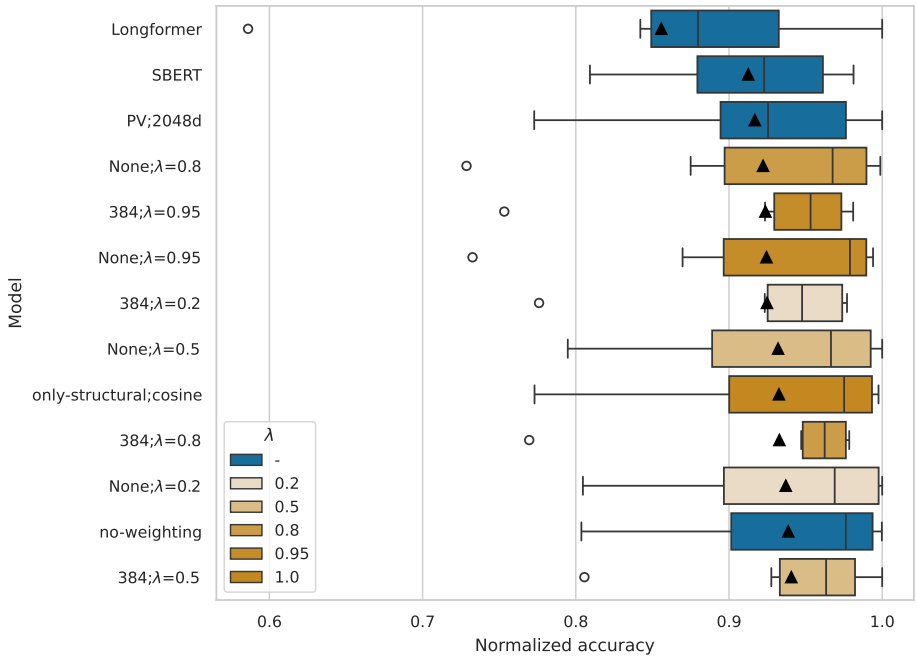

Figure 4.10: Performances of all weighting variants trained with cosine structural loss. We compare the student models to Longformer, SBERT, and only-structural;cosine.

Weighting a contextual and the max-margin MSE structural loss

Now we explore weighting hyperparameters for the max-margin MSE structural loss, DM;100d contextual teacher and S:768(ReLU)x1024(ReLU)x768;C:768 projections. Even though, this is the best performing combination for max-margin MSE structural loss, it does not surpass only-structural;mm-MSE. So, we test if different weighting of the structural and contextual loss can improve the score of the no-weighting variant. We display the models' performances in Figure 4.11. Clearly, masking out some inputs' structural loss significantly hurts performance. As we discuss in Section 4.4.1, part of the benefit of max-margin loss is that it increases the distance between different inputs' embeddings. If long inputs are masked out, the loss cannot increase the margin between their embeddings and those of the short inputs, which have not been masked out. Together with the smaller number of updates, this causes the student models to perform much worse. Note that the weighting variants with higher λ suffer considerably more.

Even without any masking, the different loss weightings fail to improve the score of only-structural;mm-MSE. We highlight that None;λ=0.5 performed a bit better than no-weighting, which are nearly identical, except that no-weighting trains with gradients twice as big. This shows that for max-margin MSE loss, the contextual does not bring any benefits. In fact, in this evaluation context, it seems that the more we train with both losses the worse is the model going to be.

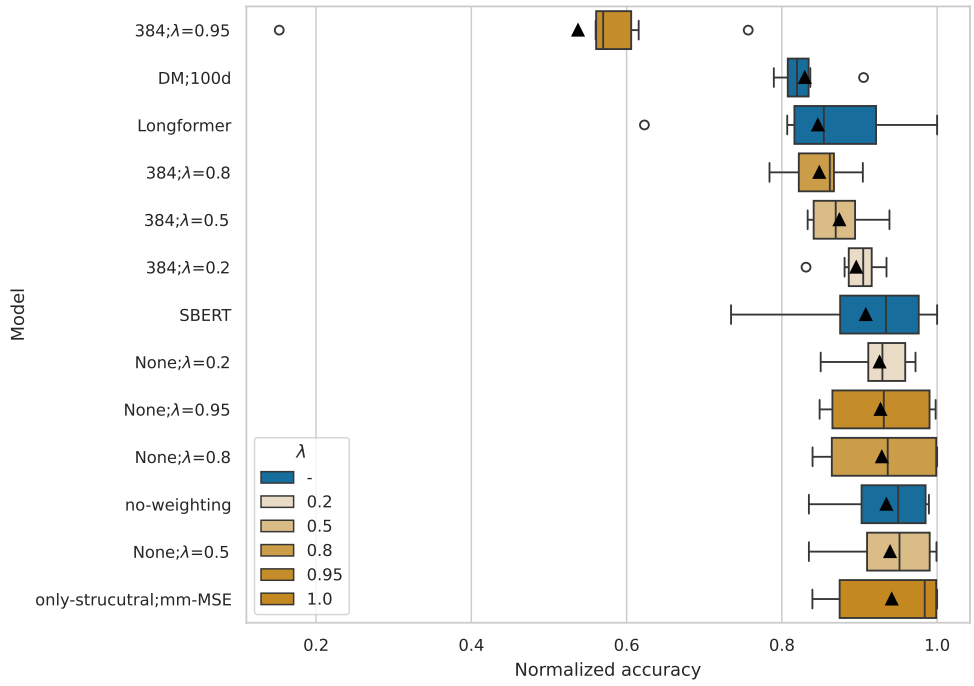

Figure 4.11: Performances of all weighting variants for max-margin MSE structural loss. We compare the students to Longformer, SBERT, DM;100d contextual teacher and only-structural;mm-MSE.

4.6 Summary

After an extensive experimentation with our teacher-student training method, we summarize what we test, but mainly how we interpret the results and what conclusions we draw from them.

First, in Section 4.4, we experiment with simple and composite structural losses. While simple losses focus only on the similarity between the student's and the corresponding teacher's embedding, composite losses also take advantage of the in-batch teacher's embedding of different inputs. Cosine is the best performing simple structural loss, surpassing even SBERT. These results show that the combination of Longformer's architecture and distillation of SBERT's embeddings can boost the student's performance even above the level of the structural teacher. The best composite loss, which is max-margin loss with MSE used as distance, performs even better. As we show in Section 4.4.1, with max-margin MSE loss, the student tries to mimic the structural teacher, while also increasing the distance between its embeddings. So the added benefit of max-margin MSE loss compared to cosine structural loss is better separation between student's embeddings.

In Section 4.5, we try to leverage the contextual loss to improve the students even further. For our contextual loss we use SoftCCA loss, to increase the correlation of the student's and the contextual teacher's embeddings projected via two feed-forward networks. After finding promising training hyperparameters for Paragraph Vector, we show that the contextual loss alone can improve Longformer's results. Yet, it cannot surpass SBERT or the student models trained with only the structural loss. This demonstrates that, in our setting, the contex-

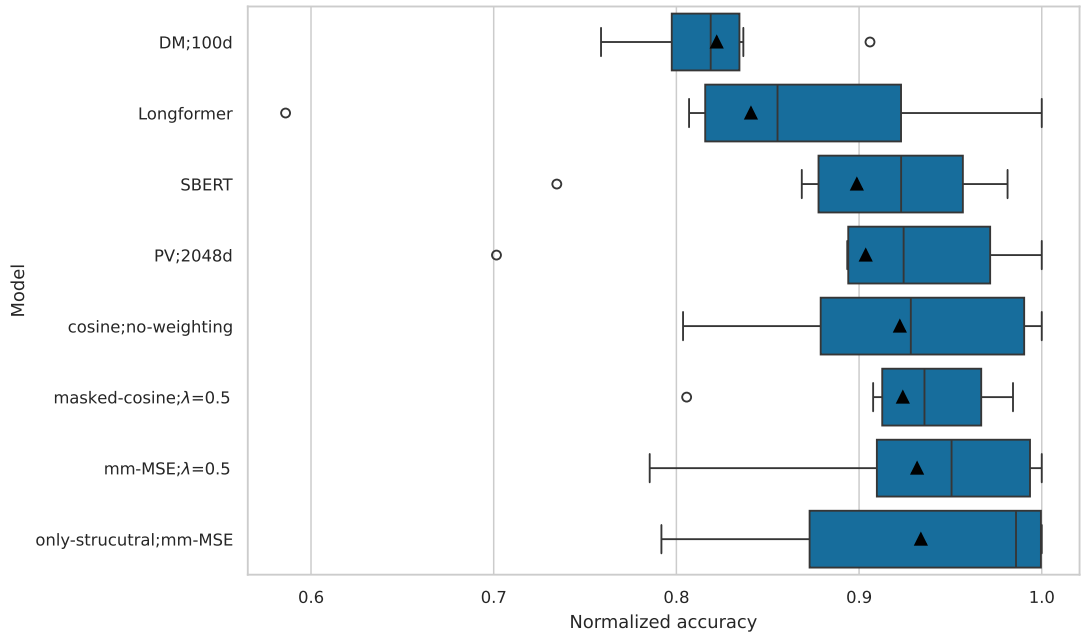

Figure 4.12: Performance of the two best student models on validation tasks per each structural loss. We compare the student's performances to Longformer and the relevant teacher models. The configuration of the student models is summarized in Table 4.10.

tual teacher is more important to the student's performance than the contextual teacher. We also find the optimal contextual loss for cosine and max-margin MSE loss. We show that the student can benefit from both cosine structural loss and the SoftCCA contextual loss, with the right projections. On the other hand, max-margin MSE loss is not as compatible with the contextual loss. Even after many trials, we fail to find projections, that would together with the max-margin MSE structural loss improve the performance of a student trained with the given structural loss only.

We also try more ways to weigh the contextual and the structural loss. In the case of max-margin MSE structural loss, we find that even with a significant emphasis on the structural loss, the student model suffers from both the contextual loss and the max-margin MSE loss used simultaneously. On the other hand, with cosine structural loss, the student model behaves intuitively. It prefers an equal balance of the structural and the contextual loss, where the structural loss is used only for inputs the structural teacher can encode whole.

To summarize, we demonstrate that, with little finetuning data, our method can improve the students embeddings with only about 2.5k training iterations. We present the performance of the best two student models on the validation tasks for each structural loss in Figure 4.12. For clarity we also include the configuration of all 4 variants in Table 4.10.

	Models	
Hyperparameter	masked-cosine;λ=0.5	cosine;no-weighting
Structural loss	cosine distance	
Contextual teacher	PV;2048d	
Student projection	768(ReLU)x4096(ReLU)x2048	
Contextual projection	-	
Weighting λ	0.5	-
Structural loss masking	longer than 386 tokens	-

(a) Models using cosine structural loss.

	Models	
Hyperparameter	mm-MSE;λ=0.5	only-structural;mm-MSE
Structural loss	max-margin MSE	
Max-margin γ	1	
Contextual teacher	DM;100d	-
Student projection	768(ReLU)x1024(ReLU)x768	-
Contextual projection	768	-
Weighting λ	0.5	-
Structural loss masking	-	-

(b) Models using max-margin MSE structural loss.

Table 4.10: Configurations of the best two models for each structural loss.

5. Evaluation

In this chapter, we evaluate the most promising configurations of our training method. We train three student models on 1M documents with the best-performing hyperparameters from Chapter 4 to show the effects of long training with our teacher-student method. We evaluate the student models on six classification and two retrieval tasks. For classification tasks, we consider three contexts, each having a different amount of available supervised data. We show how the models' performances change for each context and thus highlight the advantages and disadvantages of our training method.

5.1 Student models

We focus on the three best-performing variants of our training method from Chapter 4. More specifically, we use the hyperparameters listed in Table 4.10. However, as we train the student models and the contextual teachers on significantly more data, we label the models differently to avoid confusion. cosine-masked is trained with an equal mixture of contextual loss and cosine structural loss, which is masked out for inputs longer than the structural teacher's maximum context length. MSE-contextual is trained with an equal mixture of contextual loss and a max-margin MSE structural loss. Finally, only-MSE is trained only on the max-margin MSE structural loss. These models' hyperparameters correspond to the configurations of masked-cosine;λ=0.5, mm-MSE;λ=0.5, and only-structural;mm-MSE from Table 4.10, respectively.

5.1.1 Training data

We compile our training corpus the same as VAL-500K. Ultimately, we want to gauge the performance of our training method using the performance of the trained student models. So, to have a meaningful baseline, we train the student models only using Longformer's training data. Without any new training data, the performance of our student model is dependent only on our training method. Hence, the performance of the student models is proportional to our training method's performance.

Following Longformer's approach, we equally sample articles from the English Wikipedia[1] and documents from the RealNews dataset [Zellers et al., 2019] that have above 1200 Longformer's tokens. We label the resulting dataset as TRAIN-1M and display its statistics in Table 5.1. TRAIN-1M contains relatively long documents. The average document has around 1300 tokens, and only 34% of its documents can fit into the maximum context length of a vanilla Transformer. However, as we show in Figure 5.1, most documents have between 0 and 500 tokens or 1200 and 1700 tokens.

[1] `https://huggingface.co/datasets/wikipedia/viewer/20220301.en`

Split	Train	Validation
Documents	1 000 000	30 000
Tokens	1.37e+09	4.15e+07
Tokens per document	1375±1738	1382±1697
SBERT tokens over 384	71%	71%
SBERT tokens over 512	66%	67%

Table 5.1: Statistics of TRAIN-1M. For each split, we also show the percentage of documents with the number of SBERT tokens above the given threshold.

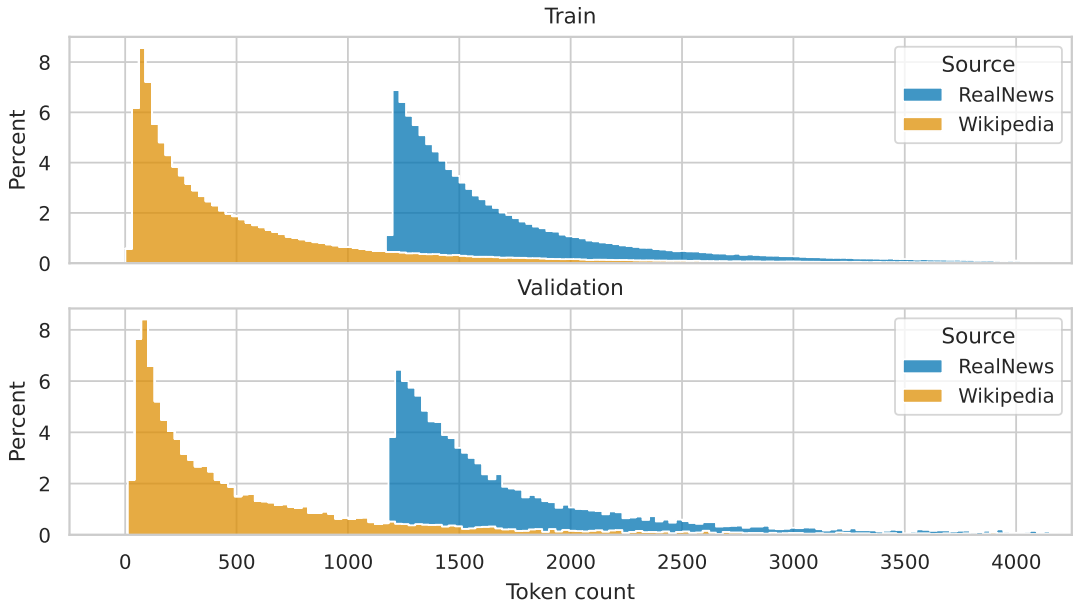

Figure 5.1: Distribution of the number of Longformer tokens per document in TRAIN-1M.

Parameter	Value
Batch size	6
Weight decay	0.1
Learning rate	3e-5
Learning rate decay	Cosine
Maximum gradient norm	1.0
Optimizer	AdamW
Gradient accumulation steps	10
Warmup update steps	500
Gradient checkpointing	Yes
Mixed-precision training	Yes

Table 5.2: Hyperparameters used for training all three student models: cosine-masked, MSE-contextual, and only-MSE.

5.1.2 Training of contextual teachers

Two of our student models use a contextual teacher. To showcase the full potential of our training method, we train the contextual teachers anew and with much more data than in the previous chapter. cosine-masked uses a 2048-dimensional Paragraph Vector [Le and Mikolov, 2014] composed of Distributed Memory and Distributed Bag of Words models. MSE-contextual uses only a 100-dimensional Distributed Memory model. cosine-masked and MSE-contextual use the best-performing hyperparameters from Section 4.5.1 labelled as PV;2048d and DM;100d respectively.

We compile the contextual teacher's training dataset in the same manner as TRAIN-1M. We avoid new data, as the contextual teachers' training data also becomes the student's training data. Therefore, we restrict the contextual teachers' training data to Longformer's training data for the reasons we mention in the previous section. However, as PV is a significantly smaller model than our student model, we can afford to train it with substantially more data. We use all available data from RealNews articles and an equal amount of Wikipedia documents. The resulting dataset has 8.3 million documents. We label the trained 2048-dimensional and 100-dimensional teachers DM and PV. To keep the models' memory footprint manageable, we restrict DM's vocabulary to 6×10^7 words and PV's vocabulary to 1.2×10^7 words. With these limitations, the models take up approximately 96GB and 124GB of memory during prediction.

5.1.3 Training of student models

Finally, we train the student models with the newly trained contextual teachers. We train on TRAIN-1M for one epoch with hyperparameters listed in Table 5.2. Thanks to the student's efficient self-attention, the models take up only 12GB of VRAM during training. We train the models on an NVIDIA A100 GPU card for approximately 30 hours.

Dataset	Inputs	Classes	Class percentage	
			Train	Test
ARXIV	documents	11	9.09±1.25%	9.09±1.30%
IMDB	documents	2	50.00±0.00%	50.00±0.00%
AAN	pairs of documents	2	50.00±1.50%	50.00±0.77%
OC	pairs of documents	2	50.00±0.07%	50.00±0.34%
PAN	pairs of documents	2	50.00±0.00%	50.00±0.00%
S2ORC	pairs of documents	2	50.00±0.09%	50.00±0.33%

Table 5.3: Overview of the classification tasks. For each task, we include the type of input classified and the mean and standard deviation of class percentages.

Dataset	Split	Documents	SBERT tokens	
			Over 384	Over 512
ARXIV	Train	28 388	100.00%	100.00%
	Test	2 500	100.00%	100.00%
IMDB	Train	25 000	24.56%	14.68%
	Test	25 000	23.54%	13.95%
AAN	Train	106 592	0.37%	0.06%
	Test	13 324	0.45%	0.08%
OC	Train	240 000	12.31%	1.07%
	Test	30 000	12.24%	1.02%
PAN	Train	17 968	69.93%	59.45%
	Test	2 906	60.82%	47.37%
S2ORC	Train	152 000	33.24%	18.67%
	Test	19 000	32.96%	18.20%

Table 5.4: Statistics of the classification tasks. We include the percentage of documents with SBERT tokens above a given threshold for each task and split.

5.2 Evaluation tasks

We thoroughly evaluate the student models using eight diverse tasks. We select six classification tasks that cover citation prediction, plagiarism detection, sentiment, and topic classification. We also include two retrieval tasks from distinct domains.

We present an overview of the selected classification tasks in Table 5.3. Besides ordinary classification tasks, we also include classifications of pairs of documents. In these tasks, the classifier bases its prediction on the comparison of two documents, or in our case, their two embeddings. As can be seen from Table 5.4, the amount of the tasks' finetuning and evaluation data ranges greatly. This becomes particularly important in Section 5.3.1, where we evaluate the student models while limiting the tasks' training data to different amounts.

We present an overview of the retrieval tasks in Table 5.5. These tasks do not have any finetuning data and test only the proximity of embeddings of similar documents. Both tasks have around 90 source articles, each with around eight similar target articles. However, GAMES has substantially more documents.

We pay special attention to the lengths of documents contained in the datasets and try to cover a span of lengths as large as possible. However, high-quality long document datasets are very rare due to their annotation's high complexity and cost. Often, a dataset is said to be composed of documents, but it contains

Dataset	Documents	Sources	Targets per source	SBERT tokens	
				Over 384	Over 512
WINES	1 662	89	8.92±1.23	100.00%	90.43%
GAMES	21 228	88	8.74±2.35	100.00%	91.78%

Table 5.5: Statistics of our similarity-based evaluation tasks. Each dataset has around 90 source documents, each similar to around nine target documents. We also include the percentage of documents with SBERT tokens above the given threshold.

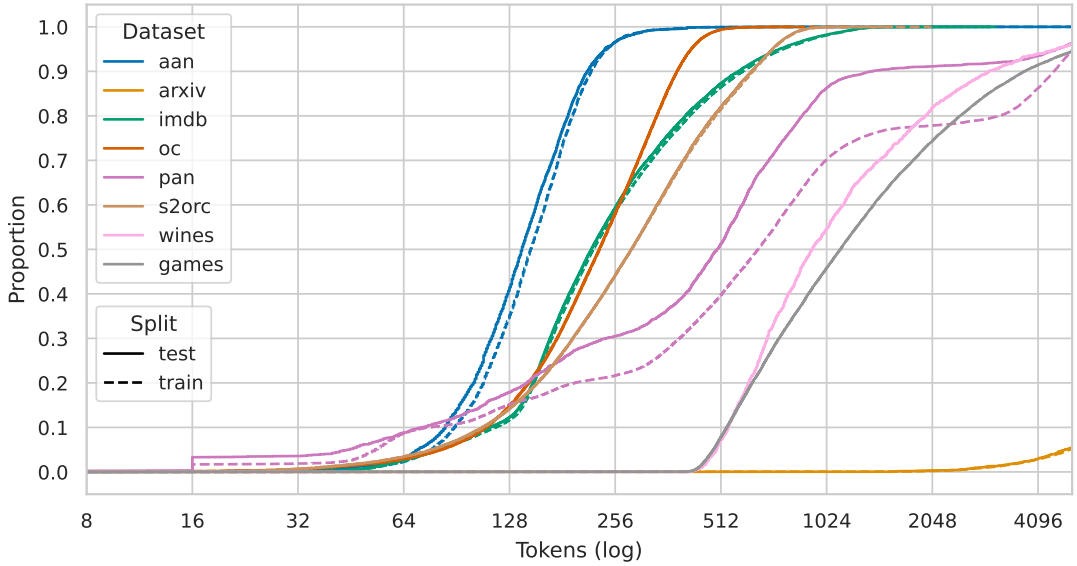

Figure 5.2: Estimated cumulative length distribution of the number of Longformer tokens in a document.

only shorter pieces of text, such as abstracts. So, we include only one dataset containing very long documents. As Figure 5.2 shows, the tasks arguably focus more on documents up to around 1024 tokens. Nonetheless, as we show in Tables 5.4 and 5.5 the tasks still contain a considerable number of documents longer than the maximum context of our structural teacher.

5.2.1 Tasks' description

IMDB movie reviews The IMDB dataset [Maas et al., 2011] (denoted as IMDB) is a binary classification dataset frequently used to evaluate long-context NLP models [Zaheer et al., 2020, Beltagy et al., 2020, Le and Mikolov, 2014]. The dataset consists of movie reviews, each with an associated rating on a 10-point scale. The reviews that rated the movie with 7 points or higher are classified as positive, and reviews with less than 4 points are classified as negative. There can be up to 30 reviews for each movie, and the test set contains a disjoint set of movies. Along with the train and test splits, the dataset contains an unsupervised split without rating or class labels.

Arxiv papers Arxiv papers [He et al., 2019] (denoted as ARXIV) is a collection of papers from ArXiv[2], an online archive of scholarly papers. Each paper contains its text truncated to 10000 words but spanning at least 1000 words. The papers are classified into 11 groups based on the scientific field of the given paper. Since each paper can be associated with several scientific fields, a small portion of the documents ($\approx 3.1\%$) appear more than once but with different labels. The scientific fields cover mainly fields of Computer science, such as Artificial intelligence or Data structures, but also fields connected with Mathematics, such as Group theory or Statistics theory.

ACL Anthology Network Corpus citations The ACL Anthology Network Corpus citations dataset [Xuhui Zhou, 2020] (denoted as AAN) is a citation prediction dataset. Each example in the dataset contains a pair of paper abstracts and is classified as positive if the first document cites the second one or negative if it does not. The dataset is compiled from the ACL Anthology Network Corpus [Radev et al., 2013], where each source paper creates positive pairs with all cited papers and a negative pair with one other randomly sampled non-cited paper.

Semantic Scholar Open Corpus citations The Semantic Scholar Open Corpus citations dataset [Xuhui Zhou, 2020] (denoted as OC) is also a citation prediction dataset in the same format as AAN. As the dataset name suggests, it was compiled from the Semantic Scholar Open Corpus [Bhagavatula et al., 2018]. In this dataset, only a single positive pair is generated for each source paper, resulting in a much higher count of unique papers compared to AAN.

PAN plagiarism detection The final classification dataset is the PAN plagiarism detection dataset [Xuhui Zhou, 2020] (denoted as PAN). It was constructed from PAN plagiarism alignment task [Potthast et al., 2013], which is a collection of pairs of web documents, where the sections relevant to plagiarism are humanly annotated both in the source as well as in the suspicious document. PAN is a binary classification task where each document pair is classified as positive or negative. Positive inputs contain the source's plagiarised section, with a part of the suspicious document containing the corresponding plagiarised section. Negative pairs are constructed from the positives by replacing the source's segment with a different part of the same document that is not annotated as being plagiarised.

Semantic Scholar Open Research Corpus citations The Semantic Scholar Open Research Corpus citations dataset [Xuhui Zhou, 2020] (denoted as S2ORC) is our third and final citation prediction dataset. The source of this dataset is the Semantic Scholar Open Research Corpus [Lo et al., 2019], where each paper is divided into sections connected via links to the papers cited within the given section. This structure is used to generate positive and negative pairs. A section is paired with an abstract of a cited paper to create a positive pair or an abstract of a non-cited paper to create a negative pair.

[2] `arxiv.org`

Parameter	Value
Hidden features	50
Hidden dropout rate	0.5
Hidden activation	ReLU
Epochs	10
Batch size	32
Weight decay	0.1
Label smoothing	0.1
Learning rate	1e-4
Learning rate decay	Cosine
Maximum gradient norm	1.0
Optimizer	AdamW
Mixed-precision training	Yes

Table 5.6: Training parameters of classification heads during evaluation.

Wines and Video games Wikipedia articles Both of our similarity-based tasks are datasets consisting of Wikipedia articles from two fields of interest: wines (denoted as WINES) and video games (denoted as GAMES) [Ginzburg et al., 2021]. Each dataset contains around 90 source articles, each associated with around nine similar articles. We find the two datasets unique as they combine two aspects that are rarely seen together. The similarities are based on expert human annotations, not proxy measures such as common citations or outgoing links. Additionally, the documents are relatively long, with around 90% of documents being longer than 512 tokens. While WINES contains fewer documents and covers fewer topics, the similarities between a source and a target document are less apparent as it is often based on a few details mentioned throughout the document.

5.3 Results

This section evaluates the student models on the previously mentioned tasks. To put the performance of the student models into context, we compare them to four baselines. First, to estimate the contribution of our training method, we compare the students to their base checkpoint, Longformer, with a mean pooling layer over its last hidden states. We also include the performances of the two contextual teachers PV and DM and the structural teacher SBERT. These models showcase the potential of our method. Ideally, we would like the student models to combine knowledge from all their teachers and surpass all of them. We evaluate all embedding models without any finetuning. With finetuning on each task, the models' performance also depends on the used finetuning method, which makes it more difficult to estimate the contribution of our training method.

As a classifier, we use a heavily regularized neural network. We train the classifier with cross-entropy loss for several epochs. We list the complete list of training hyperparameters in Table 5.6. We assess the performance of a model based on micro or binary accuracy, depending on the number of classes. We use micro-averaging for tasks with more classes to give each input the same weight. When we compare embedding models across several tasks, we use *normalized accuracy*, which we define in Section 4.2.

We evaluate the classification tasks in three rounds. In each round, we limit

the number of documents on which the classifier is trained. We find that evaluating the student models in all three contexts presents a more detailed picture of the students' performances and highlights the advantages and disadvantages of our training method. In the first round, we limit the finetuning documents to 1 thousand. With so few finetuning documents, the features that help the classifier predict the correct label must be obvious. In this setting, models that encode a few main features of their input should achieve the best results. In the second round, we increase the number of finetuning documents to 10 thousand. Finally, in the last round, we do not limit the amount of finetuning data. With more finetuning data, the classifiers can pick up on more complex features. Therefore, in this context, models that compress as much information as possible into their embedding should, in theory, achieve the best results. Contrary to the evaluations in Chapter 4, we do not limit the number of test documents. Thus, overfitting with less data should be obvious. When truncating the training splits, we downsample it following its label distribution. So, the downsampled training splits have almost equal class distribution to the original split.

For retrieval tasks, we measure the embedding's proximity with cosine distance. As we do not do any training, we use the whole dataset for evaluation. We measure the models' performances based on Mean Average Precision (*MAP*) but also present Mean Reciprocal Rank (MRR). While MAP scores the entire predicted ordering, MRR is more interpretable and can be more important in scenarios where we only care about the first positive result. When we compare models across both tasks, we use *normalized MAP*, which is computed similarly to normalized accuracy.

We show an overview of the models' performances in Table 5.7.

5.3.1 Classification tasks

We evaluate the embedding models' performance on the classification tasks in three rounds. First, we compare the overall performance of the models between the three different rounds. Then, we explore the models' performances per task in detail.

First, we focus on the overall models' performance throughout the three evaluation rounds, which we present in Figure 5.3. The relative performance of the baselines and student models is similar to what we witness in Chapter 4. In the first two rounds, the students outperform all baselines. In the last round, they beat all contextual teachers and improve the score of Longformer. However, they are only just worse than SBERT. The best student is only-MSE followed by MSE-contextual. We witness the same order of corresponding models on validation tasks at the end of Chapter 4. However, as cosine-masked is trained with the structural loss only on short inputs, it receives 3.26 times fewer update steps with the structural loss than the other two students. Consequently, cosine-masked often outperforms its contextual teacher only by a fraction, creating a noticeable performance gap between it and the other two students.

In the first round, Longformer's and SBERT's performance is underwhelming. At the same time, the best student model achieves, on average, nearly 80% of the best performance for a given task achieved by an embedding model with all finetuning data. We see this as a considerable achievement since there are 17 to

Model	ARXIV	IMDB	AAN	OC	PAN	S2ORC	Mean	Norm. mean
Longformer	.649	**.913**	.625	.889	.675	.899	.775	.894
DM	.710	.731	.570	.835	.685	.835	.728	.843
PV	**.840**	.865	.703	.891	.602	.899	.800	.923
SBERT	.819	.890	**.805**	**.935**	.640	**.944**	**.839**	**.969**
cosine-masked	.779	.843	.751	.918	.638	.925	.809	.935
MSE-contextual	.776	.842	.767	.930	.720	.942	.829	.961
only-MSE	.773	.837	.760	.929	**.740**	.941	.830	.962
10k finetuning documents								
Longformer	.508	**.892**	.521	.742	.660	.770	.682	.837
DM	.650	.699	.520	.683	.695	.697	.657	.813
PV	**.821**	.852	.537	.771	.585	.771	.723	.885
SBERT	.785	.872	.544	.787	.599	.786	.729	.893
cosine-masked	.747	.821	.568	.865	.629	.868	.750	.918
MSE-contextual	.752	.826	**.630**	.886	.702	.901	.783	**.963**
only-MSE	.748	.821	.619	**.892**	**.717**	**.904**	**.784**	**.963**
1k finetuning documents								
Longformer	.252	**.835**	.509	.655	.602	.677	.588	.814
DM	.215	.591	.508	.567	.675	.581	.523	.735
PV	.640	.779	.511	.654	**.677**	.661	.654	.918
SBERT	.606	.780	.514	.601	.565	.605	.612	.860
cosine-masked	.584	.726	.529	.747	.658	.703	.658	.922
MSE-contextual	**.645**	.762	.541	**.770**	.629	.733	**.680**	**.953**
only-MSE	.642	.746	**.545**	.763	.635	**.750**	**.680**	**.953**

(a) Classification tasks

Model	GAMES	WINES	Mean	Norm. mean
Longformer	.158	.096	.127	.724
DM	.130	.115	.123	.717
PV	.173	.133	.153	.887
SBERT	.191	.143	.167	.964
cosine-masked	.165	**.148**	.157	.917
MSE-contextual	.186	.145	.165	.957
only-MSE	**.198**	.145	**.172**	**.989**

(b) Retrieval tasks

Table 5.7: Performance of evaluated embedding models on all downstream tasks. For classification tasks, we show the performance on all three rounds, revealing the performance with all finetuning data first, then with them being limited to 1k and 10k documents. For classification tasks, we show binary or micro accuracy. For retrieval tasks, we show MAP. We also display the mean score and the mean of normalized scores for each model.

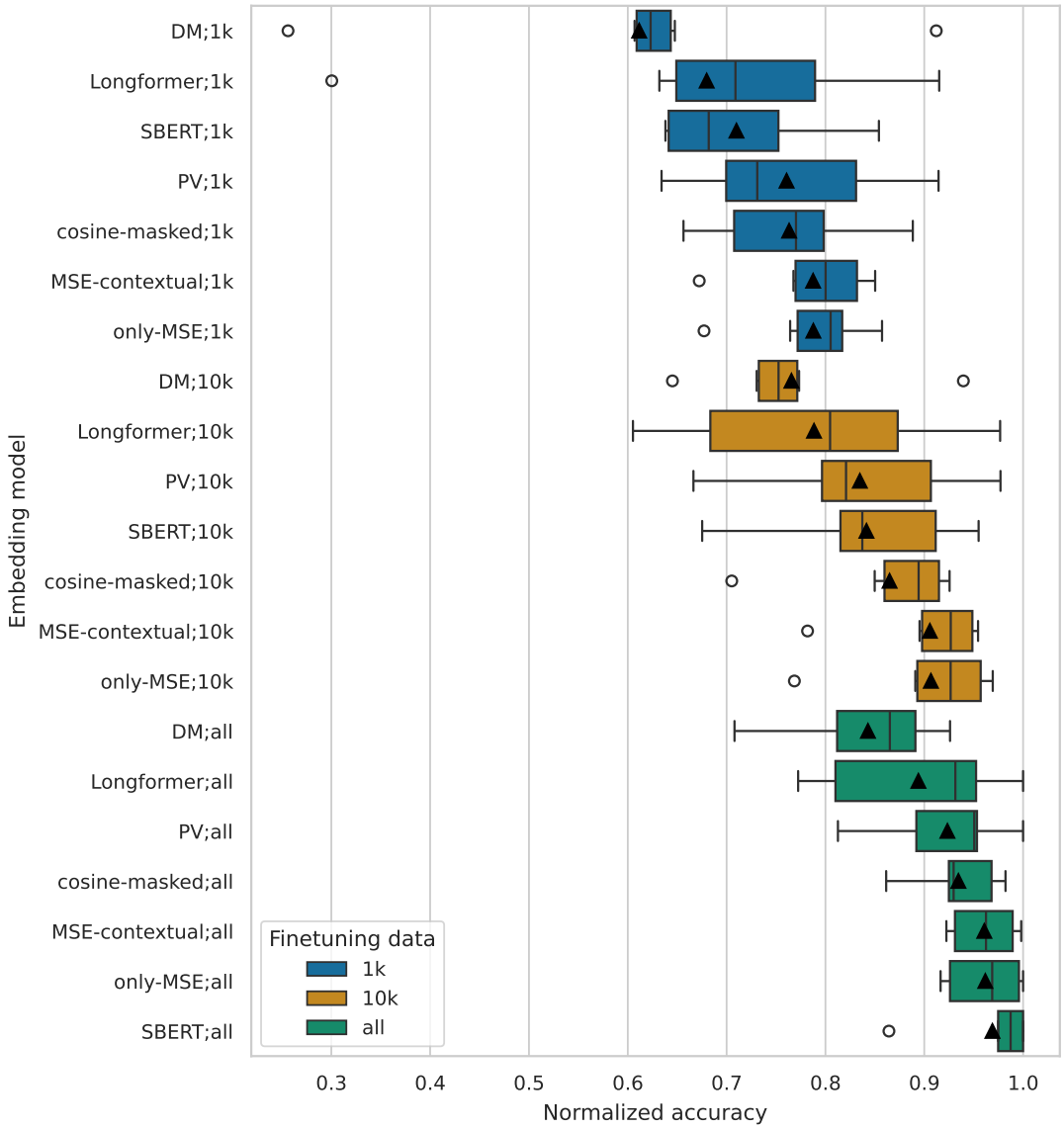

Figure 5.3: Overall relative performance of embedding models throughout the three rounds. In the first two rounds, we limit the number of finetuning documents to 1k and 10k, but do not set any limit in the third round labeled as "all".

240 times more finetuning documents in the last round compared to the first one. In the second round, the classifiers are trained on 10k finetuning documents, and the performances naturally increase. Particularly for SBERT, as it now surpasses PV, which improves by a relatively small amount. These effects are also noticeable in the performance of cosine-masked as there is a more significant gap between it and the other two students that rely more on SBERT. Finally, with all finetuning data, the differences between models' performances diminish as the best models improve only marginally. SBERT takes the greatest advantage of the increase in finetuning data and surpasses all other models. We see this as a demonstration that the embedding model does not strictly need a large maximum context for the selected set of tasks. In other words, despite the moderately large documents, we can reach a competitive performance by only considering the first 384 tokens of each input. This is apparent, especially for ARXIV, where we can imagine classifying the field of a scholarly paper based on just its abstract. So, as the context is of minor importance, SBERT benefits from its full attention and surpasses the best student by a small fraction. Nonetheless, all student models can improve the scores of their contextual teachers and base checkpoints. Moreover, the students reach consistent and comparable performances despite being trained differently. cosine-masked trains with a different contextual teacher, contextual loss, structural loss, and weighting of the two losses than the rest of the students. MSE-contextual trains with a significantly less performant contextual teacher, while only-MSE does not use a contextual teacher. This shows that our method is robust and open to multiple changes in various aspects.

We now examine the models' performances per each task, which we plot in Figure 5.4. The relative models' performances on a given task stay consistent throughout the three rounds, so we focus only on the last two. With 10k finetuning documents, the students perform best on classifications of document pairs. On IMDB, Longformer shows the best performance, which is surprising given it is the second-worst model in both rounds. On ARXIV, PV shows the best performance as it benefits from its large embedding and unlimited context. Arguably, both of these attributes play a role, as DM with the same unlimited context but with more than 50 times smaller embedding performs admirably well but worse than PV. As we increase the number of finetuning documents, SBERT substantially improves. We register the largest performance increase for tasks with the most finetuning data available. These are AAN, OC, and S2ORC. We also highlight SBERT's performance on ARXIV in both rounds, where it outperforms all students with eight times larger maximum context and comes relatively close to the performance of PV. This demonstrates that, despite the task being composed of only very long documents, we can achieve a competitive performance based on just the first few hundred tokens. The insignificance of an embedding model's lack of context may partly explain why the students' performances for a given task are not affected by the length of the tasks' documents. For example, on tasks with longer documents such as ARXIV or PAN, cosine-masked and MSE-contextual perform on par with only-MSE despite being trained with a contextual teacher, which should theoretically provide the students with a larger context.

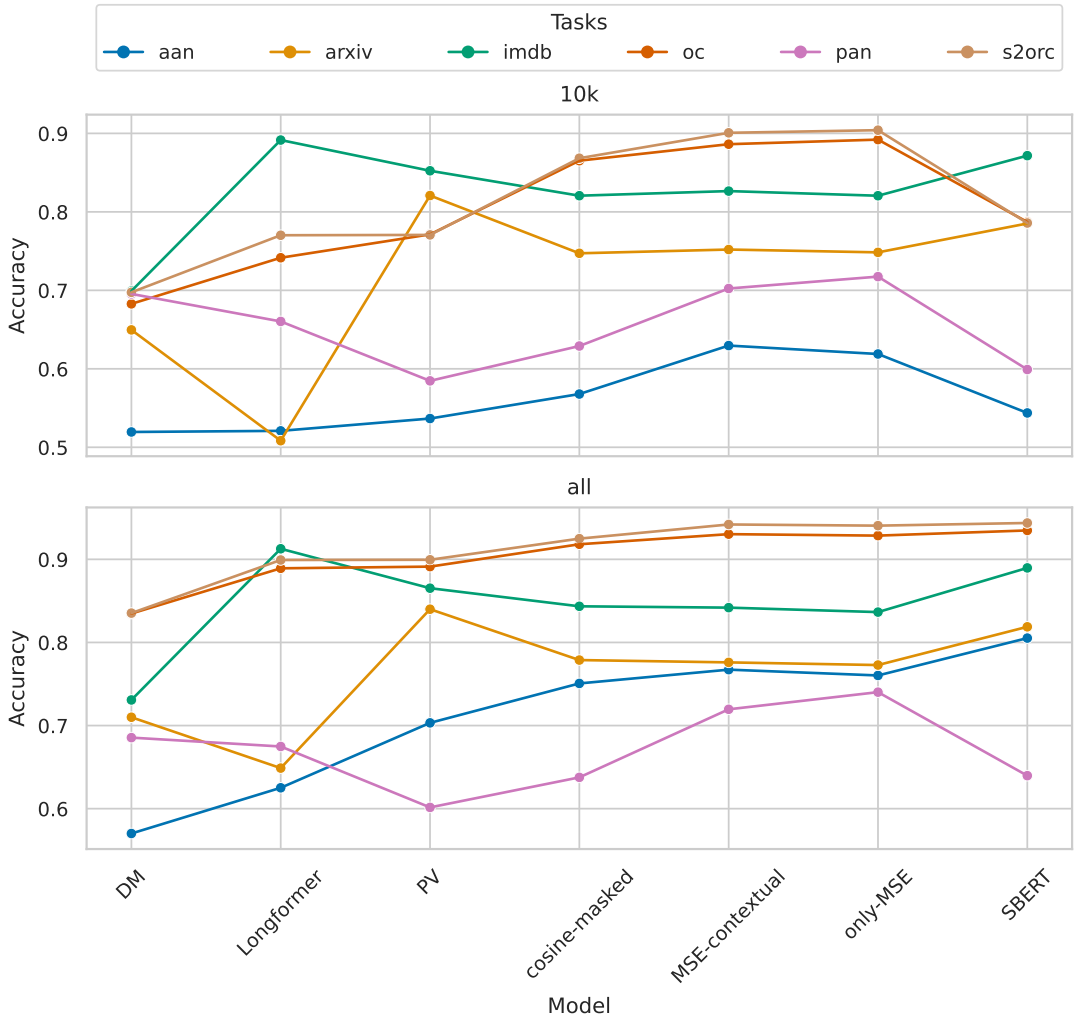

Figure 5.4: Performance of embedding models on evaluation tasks with and without (labeled as "all") the train split limited to 10k documents.

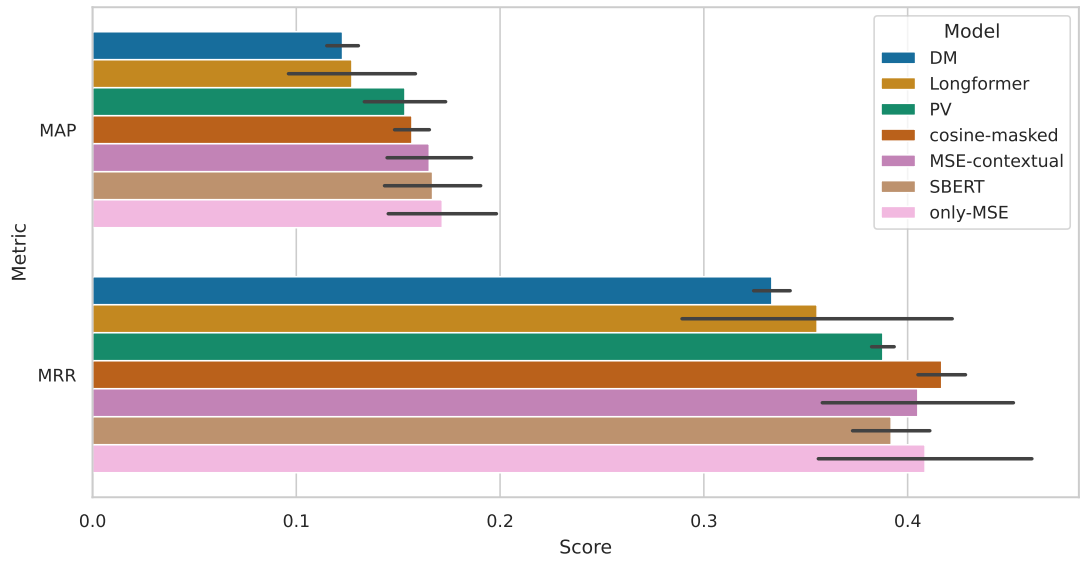

Figure 5.5: Performance of models on retrieval tasks. We mark the mean score with a bar and the span between each task's score with an error bar.

5.3.2 Retrieval tasks

We plot the overall models' performances on the retrieval tasks in Figure 5.5. In terms of MAP, the best-performing model is only-MSE closely followed by SBERT and the other two student models. For Mean Reciprocal Rank, the best model is cosine-masked, one of the most consistent models in both metrics. This suggests that using a structural teacher only for inputs, which it can process as a whole, may lead to a more consistent student model.

We also plot the models' performances per each task in Figure 5.6. As the results suggest, GAMES is an easier task than WINES. This may be unexpected since, compared to WINES, GAMES has more total documents but a similar amount of source and target documents. Consequently, GAMES contains much more "noise" documents, which may hurt the performance. However, as we mention in the tasks' description, the selection of topics for GAMES is much wider, and the differences between documents are far less nuanced. On the other hand, the similarities of documents in WINES are sometimes based on a few details mentioned throughout the document.

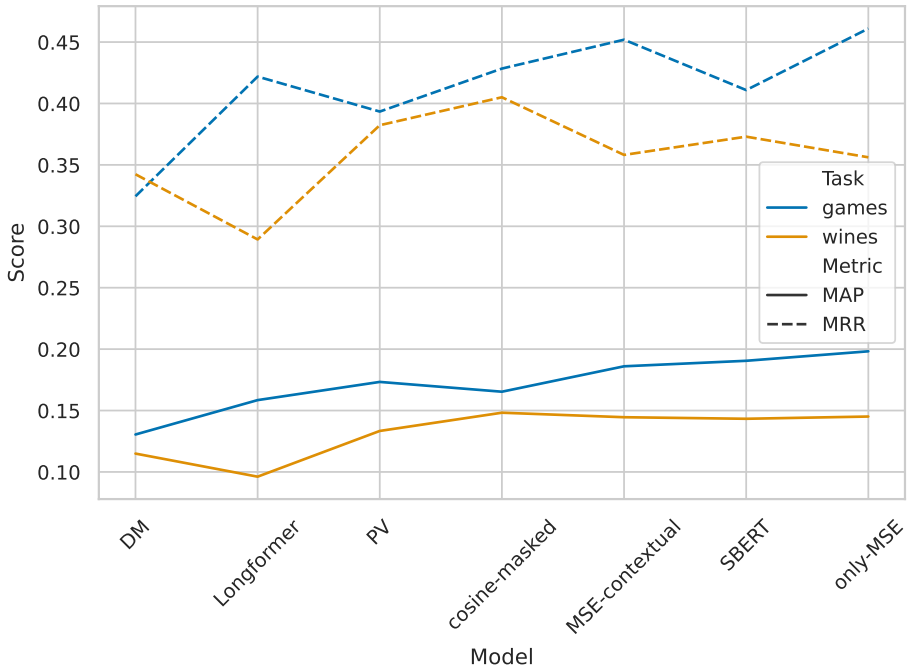

Figure 5.6: Performance of embedding models on each retrieval task.

Conclusion

In pursuit of our goal to train a Transformer document embedder with low computational resources and unsupervised text corpora, we use a teacher-student training approach. We combine the qualities of two distinct embedding models and distill their embeddings into a single student model. For the two teachers we choose SBERT [Reimers and Gurevych, 2019] for its capacity to model complex text structure and Paragraph Vector [Le and Mikolov, 2014] for its unlimited context length. We label the teachers as *structural* and *contextual* respectively. For our student model we choose an efficient transformer with sparse attention that is still able to capture text structure, but has longer context length than a vanilla Transformer. In our case we initialize our student model with Longformer [Beltagy et al., 2020], though as our technique does not rely on any specific Longformer's features, it can be theoretically applied to any other Transformer with sparse attention such as BigBird [Zaheer et al., 2020].

We train the student model on a mixture of two losses, each corresponding to one teacher. We choose the structural loss to enforce exact similarity with the structural teacher's embeddings. We train several losses, but obtain the best results with max-marginals Mean Squared Error (*MSE*) loss. For the contextual loss we use a variant of Canonical Correlation Analysis (*CCA*) [Hotelling, 1992] called SoftCCA [Chen et al., 2016]. SoftCCA forces correlation between the student's and the contextual teacher's embeddings projected via two separate feed-forward networks. While the contextual loss alone can improve Longformer's performance, we find the performance gain not as significant as with the structural loss. However, the student benefits from training on both losses simultaneously.

We evaluate ...

5.4 Future work

Bibliography

Waleed Ammar, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, Ahmed Elgohary, Sergey Feldman, Vu Ha, et al. Construction of the literature graph in semantic scholar. *arXiv preprint arXiv:1805.02262*, 2018.

Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *International conference on machine learning*, pages 1247–1255. PMLR, 2013.

Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Chandra Bhagavatula, Sergey Feldman, Russell Power, and Waleed Ammar. Content-based citation recommendation. *arXiv preprint arXiv:1802.08301*, 2018.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Xiaobin Chang, Tao Xiang, and Timothy M Hospedales. Scalable and effective deep cca via soft decorrelation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1488–1497, 2018.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S Weld. Specter: Document-level representation learning using citation-informed transformers. *arXiv preprint arXiv:2004.07180*, 2020.

Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*, 2021.

Dvir Ginzburg, Itzik Malkiel, Oren Barkan, Avi Caciularu, and Noam Koenigstein. Self-supervised document similarity ranking via contextualized language models and hierarchical inference. *arXiv preprint arXiv:2106.01186*, 2021.

Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

Jun He, Liqun Wang, Liu Liu, Jiao Feng, and Hao Wu. Long document classification from local word glimpses via recurrent attention learning. *IEEE Access*, 7:40707–40718, 2019. doi: 10.1109/ACCESS.2019.2907992.

Harold Hotelling. Relations between two sets of variates. In *Breakthroughs in statistics: methodology and distribution*, pages 162–190. Springer, 1992.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of bert. *arXiv preprint arXiv:1908.08593*, 2019.

Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*, 2016.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

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. *arXiv preprint arXiv:1907.11692*, 2019.

Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Dan S Weld. S2orc: The semantic scholar open research corpus. *arXiv preprint arXiv:1911.02782*, 2019.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

Malte Ostendorff, Nils Rethmeier, Isabelle Augenstein, Bela Gipp, and Georg Rehm. Neighborhood contrastive learning for scientific document representations with citation embeddings. *arXiv preprint arXiv:2202.06671*, 2022.

Martin Potthast, Matthias Hagen, Tim Gollub, Martin Tippmann, Johannes Kiesel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. Overview of the 5th international competition on plagiarism detection. In *CLEF Conference on Multilingual and Multimodal Information Access Evaluation*, pages 301–331. CELCT, 2013.

Markus N Rabe and Charles Staats. Self-attention does not need $O(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

Dragomir R Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. The acl anthology network corpus. *Language Resources and Evaluation*, 47:919–944, 2013.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Nils Reimers and Iryna Gurevych. Making monolingual sentence embeddings multilingual using knowledge distillation. *arXiv preprint arXiv:2004.09813*, 2020.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Amanpreet Singh, Mike D'Arcy, Arman Cohan, Doug Downey, and Sergey Feldman. Scirepeval: A multi-format benchmark for scientific document representations. *arXiv preprint arXiv:2211.13308*, 2022.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems*, 33:16857–16867, 2020.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers, 2019.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6), dec 2022. ISSN 0360-0300. doi: 10.1145/3530811. URL `https://doi.org/10.1145/3530811`.

Trieu H Trinh and Quoc V Le. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*, 2018.

Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 12, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Weiran Wang, Raman Arora, Karen Livescu, and Jeff A Bilmes. Unsupervised learning of acoustic features via deep canonical correlation analysis. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4590–4594. IEEE, 2015.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective long-context scaling of foundation models. *arXiv preprint arXiv:2309.16039*, 2023.

Noah A. Smith Xuhui Zhou, Nikolaos Pappas. Multilevel text alignment with cross-document attention. In *EMNLP*, 2020.

Liu Yang, Mingyang Zhang, Cheng Li, Michael Bendersky, and Marc Najork. Beyond 512 tokens: Siamese multi-depth transformer-based hierarchical encoder for long-form document matching. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1725–1734, 2020.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Hamed Zamani, Mostafa Dehghani, W Bruce Croft, Erik Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 497–506, 2018.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. *Advances in neural information processing systems*, 32, 2019.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

List of Figures

List of Tables