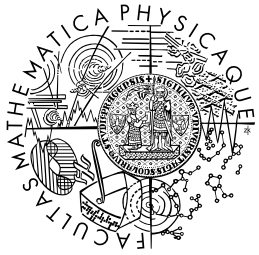


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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

- what we are aiming to do
- why is it important/useful — where are embeddings used
- long documents — why?
- transformers — why?

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 structural and contextual 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

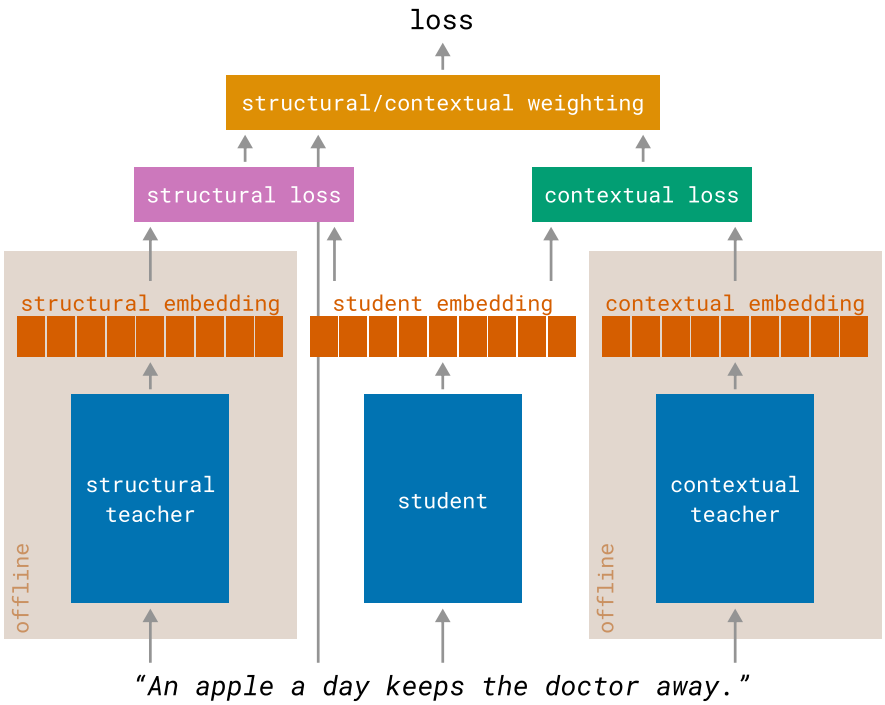


Figure 3.1: Architecture of our teacher-student training. We distill the qualities of the teachers' embeddings through corresponding losses into a student model. Since we do not update the weights of either teacher, generation of their embeddings can be done offline, before training.

the structural teacher's embedding $y_{\mathcal{T}_S}$ and the contextual teacher's embedding $y_{\mathcal{T}_C}$, 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 be dependent 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.

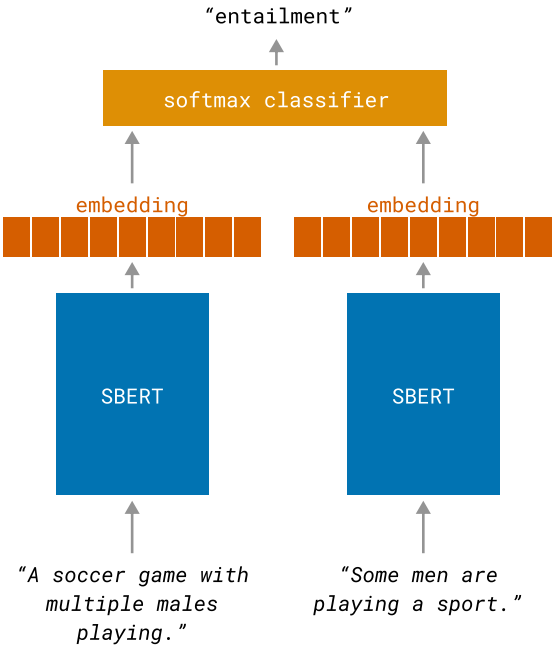


Figure 3.2: Siamese network architecture used to train SBERT. The pair of sentences from a NLI dataset are classified into three classes "entailment", "netural" and "contradiction".

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 Vector can produce embeddings even for long documents, but it processes text very shallowly, which prohibits understanding any complex structures. We hope to synthesize both of these 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 fine-tuned with Natural Language Inference (NLI) datasets to produce semantically meaningful embeddings. We show an illustration of 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].

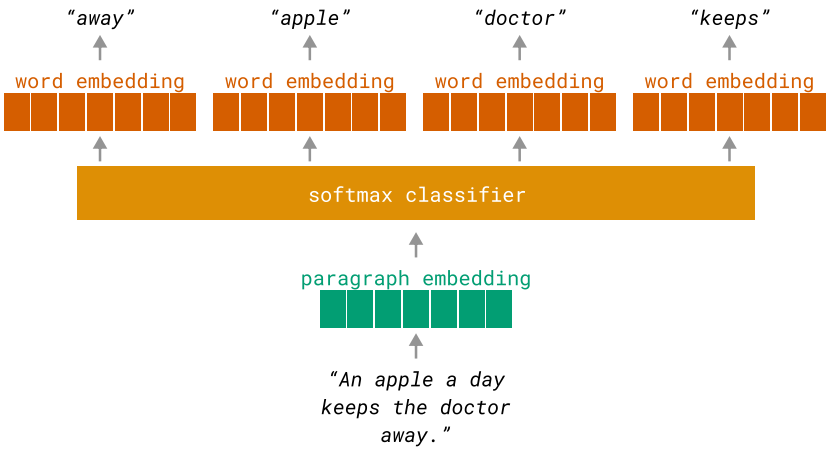


Figure 3.3: Architecture of Distributed Bag of Words. The model predicts words from a document, only using the document's embedding.

3.2.2 Paragraph Vector

Paragraph Vector [Le and Mikolov, 2014] (also known 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 using both and concatenating their embeddings. 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 embedding of the whole paragraph. On the other hand, DM, whose architecture is depicted in Figure 3.4, additionally uses the embeddings of the surrounding words within 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' embeddings. We choose the student's architecture to be at a midpoint between the

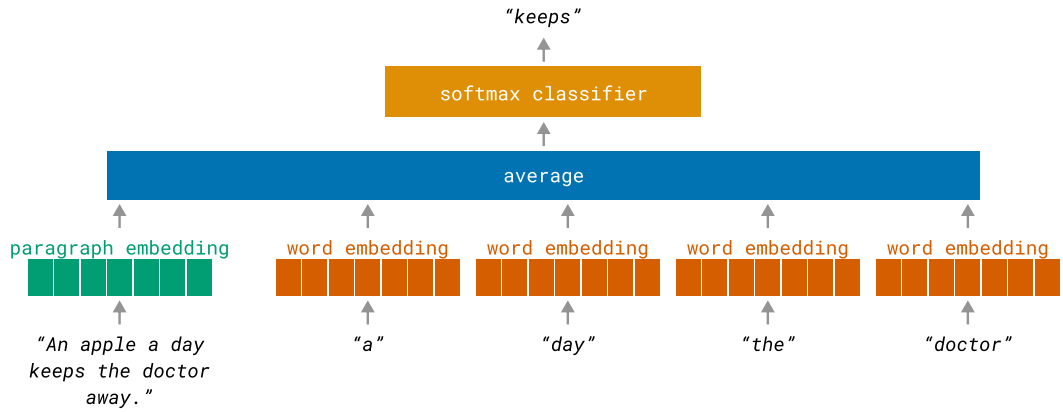


Figure 3.4: Distributed Memory model architecture. The model predicts the input words' neighboring word for the input paragraph.

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 since the Transformer is a well-tested architecture. 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 a pre-trained Transformer model with a sparse attention mechanism 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 and 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. However, there is an alternative[1] that tries to mimic the dataset from the original paper.

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

4. Experiments

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

This chapter is laid out as follows. First we describe the training data in Section 4.1. Second we discuss how we compare models in Section 4.2. Third we 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.

4.1 Training data

We create the training dataset for our validation, labelled as `val_500k`, similarly to how we put together the final training dataset `train_1500k` in Chapter 5. In short, we equally sample documents from English Wikipedia and RealNews articles [Zellers et al., 2019], which are at least 1200 Longformer tokens long. We use a subset of Longormer's training data, to make the comparison between a learned student model and its base checkpoint fair. In this way, the trained student model will not see any new data compared to Longformer. Thus any performance gains of the student model over its base checkpoint, can be attributed to our training method, rather than to a higher-quality training dataset.

We show some of the statistics of `val_500k` in Table 4.1. Very similar to `train_1500k`, `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 document's length distribution in Figure 4.1. As Wikipedia contains relatively short documents, while RealNews does not contain document below 1200 tokens, the source's distributions are well spaced out.

4.2 Validation tasks

We compare the embedding models based on their performance on a set of downstream tasks. We use a subset of our evaluation tasks, which we describe in

Split	Train	Validation
Documents	500 000	10 000
Tokens	6.85e+08	1.37e+07
Tokens per document	1370.74±1723.38	1371.65±1717.24
SBERT tokens over 384	70.56%	70.07%
SBERT tokens over 512	66.37%	66.14%

Table 4.1: Statistics of `val_500k`. For each split we also show the percentage of documents with the number of SBERT tokens above given threshold.

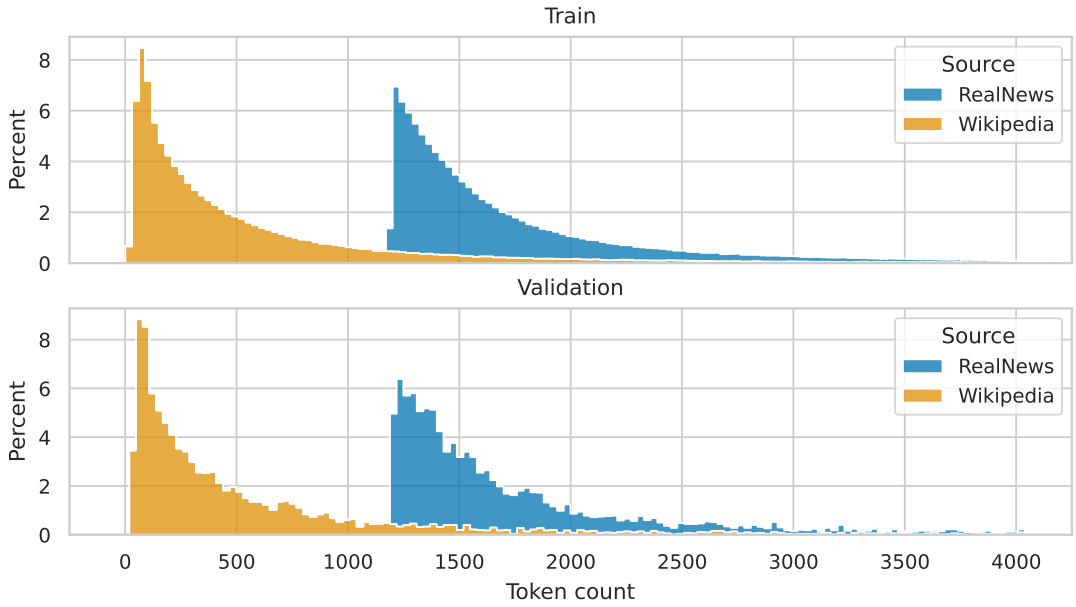


Figure 4.1: Distribution of train and validation documents' lengths for `val_500k`.

Chapter 5. We include tasks that either have large enough training split suitable for cross-validation or a validation split. All tasks are classifications that 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 according to their label distribution, so that the truncated split has nearly identical label distribution to the original one. We present the validation tasks along with their document count in Table 4.2.

We use binary or micro-averaged accuracy as the scoring metric. Often we need to compare performance across several tasks. However not all tasks are equally difficult and so averaging accuracies would lead us to favoring models that performed well on easy tasks, and undervaluing models that performed well on the more difficult tasks. Therefore, we normalize the accuracy by the highest score reached for the given task within the considered models and thereby making the tasks equally difficult. We call this metric *normalized accuracy*.

When validating a trained embedding model on a task we finetune a head that transforms the embeddings into the output format required by the given task. We do not finetune the embedding model itself. Besides speeding up the validation, this gives us also more genuine picture of the performance of the embedding model. 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 classifier's hyperparameters and training parameters in Table 4.3.

4.3 Student's model configuration and baselines

As we explained in Section 3.3, we chose Longformer [Beltagy et al., 2020] as our student model. We use Longformer's base version with about 126M parameters

Dataset	Documents		Classes	Class percentage	
	Train	Test		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 for comparing embedding models in this chapter. We truncated splits marked with † to speed up validation of models. We truncate a split by downsampling it according to its label distribution. The class distributions of all tasks are well balanced except for AAN as can be seen from the mean and standard deviation of class percentages for given task and split.

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: Training hyperparameters used for training classification heads during validation.

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 used for all student's trainings in this chapter.

implemented by HuggingFace `transformers` library[1].

To obtain the input's embedding we average the last layer's hidden states. We do not use global attention and restricting ourselves to only sliding window attention, with the window sizes ω set to the default 512 tokens. We also experimented with obtaining the document embedding from the last layer's hidden state of the `[CLS]` token, while simultaneously setting the global attention for the same token. However, this configuration resulted in worse performance and thus we continue using mean pooling of the last layer's hidden states and no global attention.

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

4.3.1 Baselines

As we already mentioned in the beginning of this chapter, the goal of the following experiments is to surpass both teachers as well as the base checkpoint of our student model using our teacher-student training approach. To that end we compare our student model throughout this chapter to three models: Longformer [Beltagy et al., 2020], SBERT [Reimers and Gurevych, 2019], and PV [Le and Mikolov, 2014]. Longformer is the base checkpoint of our student model. By comparing the student model to Longformer, we judge how much our training method improves document embeddings. As we mentioned above, we use a subset of Longformer pre-training data to eliminate the option that the student model gains performance just because of the new training data. For context, we train the student models for only 3.8% iterations of Longformer's pre-training with 8-times smaller batch size.

We compare the student models to the two teachers; SBERT and PV. We hope to surpass both teachers by combining their respective qualities and the architectural advantages of the student model over the teachers. Compared to SBERT, the student model has maximal context 8 times longer, whereas compared to PV

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

it has much more parameters and therefore should be able to learn more complex structural relationships within the input. And so, by comparing the student model to the teachers we estimate how well our method distills the qualities of the teacher's embeddings into the student's. We discuss the configuration and checkpoint of the structural teacher in the following sections. Paragraph Vector's training is a part of our teacher-student training method, and so we discuss further details in Section 4.5.1, where we experiment with PV's training hyperparameters. Until then we compare the student model only to the structural teacher.

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 and 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 essential to the performance of the student model than the contextual quality. Note that we arrive to the same conclusions later in this chapter in Section 4.5.2. Therefore, we prioritize finding the best performing hyperparameters of the structural loss first and adapting the contextual loss to it, after.

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 relatively small model with above average performance[2]. We use SBERT's implementation from HuggingFace `transformers` library[3] with mean pooling layer above the last layer's hidden states. We do not perform any finetuning and use the pre-trained weights only.

As we explain in Section 3.1.2, we chose structural loss to be more restrictive, forcing an exact similarity between the two embeddings. Therefore, we try two different exact losses: Mean Squared Error (*MSE*) and cosine distance. We chose 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. We compare the student's performance to the relevant baselines: Longformer and SBERT. As we show in Figure 4.2, with cosine distance the student model manages to surpass both baselines. On the other hand, using MSE as a structural loss has damaging effects on the student's performance. Notably, SBERT's scores are significantly higher than Longformer's despite its much shorter context length. That is to show, that unless the student's embeddings are finetuned, its longer context length is of little benefit.

To summarize, going forward we use cosine loss as the structural loss.

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

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

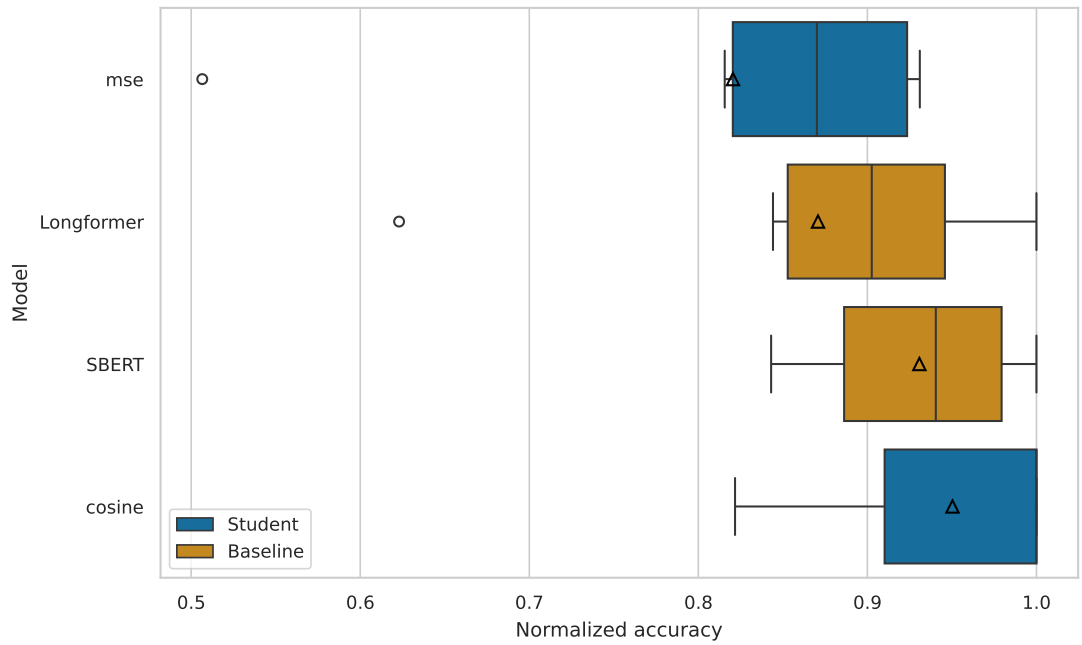


Figure 4.2: Boxplot of normalized accuracies on validation tasks of baselines and student models trained with structural teacher. We mark the normalized accuracy means per model by a triangle.

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 max-marginals or contrastive loss. We call such losses *composite* as opposed to the *simple* losses, such as MSE or cosine distance. In composite losses, the positive component compares the student's and the corresponding teacher's embedding and rewards the model if they are similar. The negative component compares the student's and the non-corresponding teacher's embeddings and punishes the model if they are similar. For brevity, we call the corresponding teacher's embedding *positive*, while the non-corresponding ones *negatives*.

Our motivation behind the composite losses was that the negatives could give the model a more precise indication where it should move its embeddings in the vector space and therefore it could converge faster to the structural teacher's embeddings. However, as we discuss in further subsections, this is not what actually happens. Nevertheless, some of the models performed so well we do not want to overlook these losses. Additionally, in our opinion, these experiments are a solid ground for a future research.

We explored two types of composite losses: max-marginals and contrastive. Both losses maximize the distance to the negatives while minimizing the distance to the positive, but each does it differently, which is best shown by a formula. 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 formulate the max-marginals loss in Equation 4.1 and the contrastive loss in Equation 4.2.

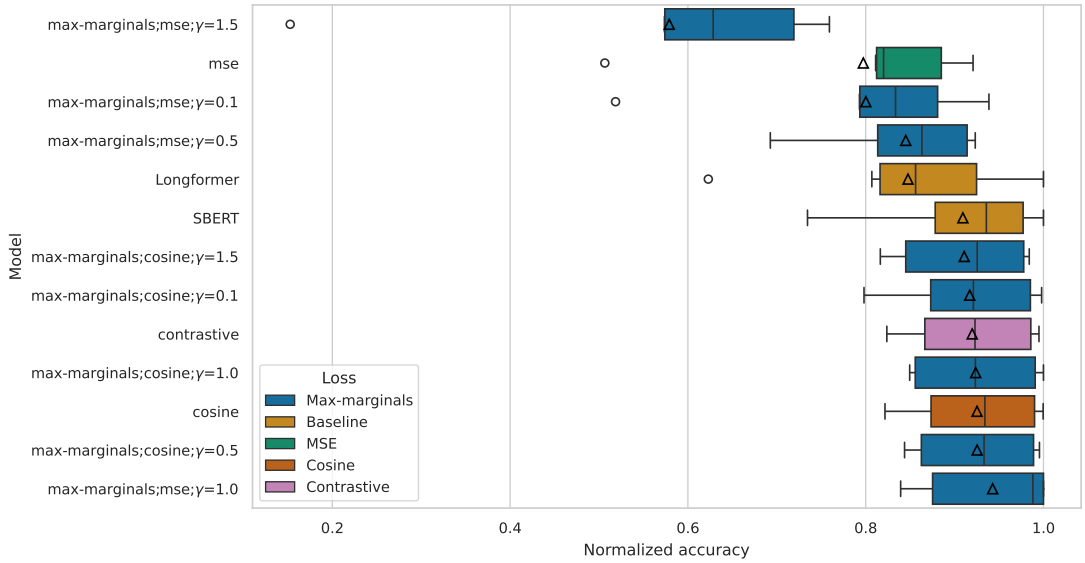


Figure 4.3: Comparison of student model performances by loss type.

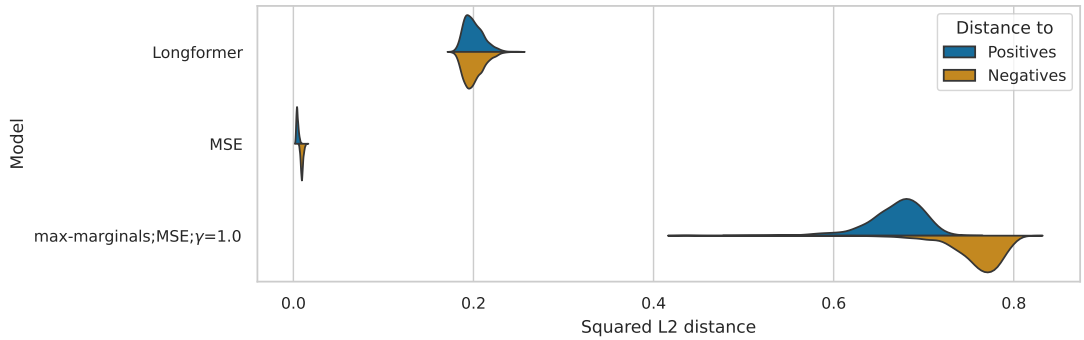
$$\mathcal{L}_{\text{max-marginals}}(y, y_{\text{pos}}, Y_{\text{neg}}) = \text{sim}(y, y_{\text{pos}}) - \gamma \frac{1}{|Y_{\text{neg}}|} \sum_{y_{\text{neg}} \in Y_{\text{neg}}} \text{sim}(y, y_{\text{neg}}) \tag{4.1}$$

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

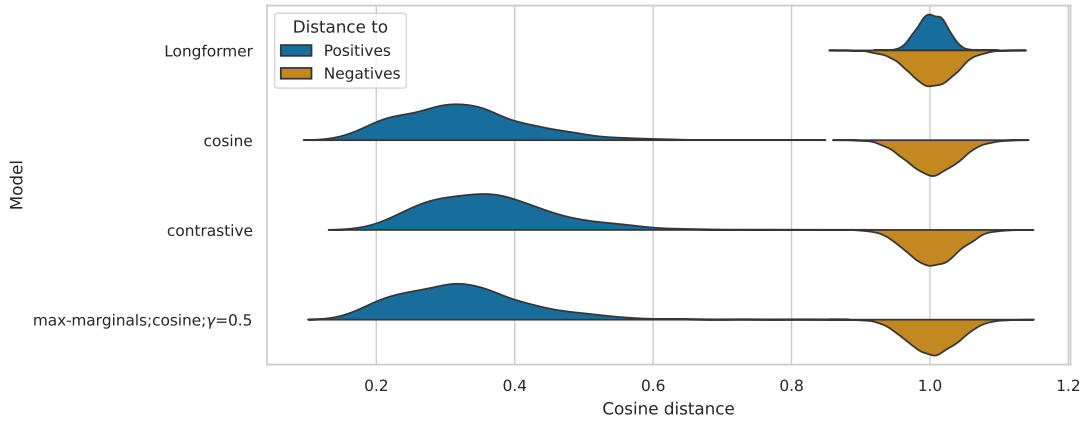
For max-marginals loss, we train with both MSE and cosine distance as sim and simultaneously try several weightings γ. We compare models trained with the composite losses with those trained with the simple losses in Figure 4.3. Composite losses using MSE, seem to benefit from the negative loss component as 3 out of 4 outperformed the simple MSE loss. However, for only one version, the benefit is large enough to outperform the two baselines. With cosine, it seems that the negative loss component hurts the performance since only one version outperformed the simple cosine loss and it did so by only 10^{-4}. To explain the differences between behaviours of MSE and cosine composite losses we carry out an analysis in the subsequent section.

Analysis of composite losses

The results of composite losses presented in Figure 4.3 suggest, that the composite losses using MSE benefit from the negative loss component, whereas the ones using cosine do not. Furthermore, for most MSE max-marginals losses the benefit of negative loss component is only marginal. To explain these results we compare the distances to positives and negatives for several chosen models in Figure 4.4. Composite losses using MSE widen the gap between positive and negative distances at the expense of increasing the distance to the positives. And so, the benefit of the negative component is not that it gives a more precise training signal and causes a faster convergence to the structural teacher's embeddings. Conversely, the benefit comes from a clearer separation between embeddings of



(a) With MSE



(b) With cosine

Figure 4.4: Distribution to distances between the model's and the structural teacher's embeddings. A distance to teacher's embedding of the same document is labelled as *positive*, whereas distances to teacher's embeddings of another document is labelled as *negative*. The distances were generated from the first 1000 documents of `val_500k`'s validation split.

different documents, even if the generated embeddings do not correspond to the teacher's embeddings that well. The situation is different for cosine, since its range is limited from the top and therefore the only option how to increase the gap between the positives and the negatives is to decrease the distances to the positive. Additionally, all of the negatives are almost perpendicular to the generated embedding and therefore direct the generated embedding in only one dimension out of the 768. Consequently, the benefit of the negative component in cosine composite losses is very small.

In summary, contrary to our initial hypothesis, the negative loss component does not ensure faster convergence of the student's embeddings to those of the structural teacher. In fact, the negative component causes the student's embedding to move away from the structural teacher's embedding and hence, goes against the goal of our training. Nonetheless, as we did not want to throw away a well-performing model and were interested in how the composite loss would cooperate with a contextual loss, we continue experimenting the best performing composite loss in the following sections. However, pure cosine loss remains the preferred structural loss, since it is consistent with the goal of our training.

4.5 Structural and contextual loss

In the previous section, we focused on structural losses and found the best performing one. In this section, we experiment with contextual loss and search for the one that will complement the already chosen structural loss the best. We also search for a contextual loss that complements the best performing composite loss, which we explored in Section 4.4.1. Despite contradicting the goal of our training, the max-marginals loss with MSE outperforms all other structural losses and therefore we continue experimenting with it in this section.

In Section 3.2.2 we explained why we use Paragraph Vector [Le and Mikolov, 2014] as our contextual teacher. Since there is no concept of a pre-trained PV, as in the case of transformers, we train PV from scratch. First, in Section 4.5.1 we explore the hyperparameters of Paragraph Vector's training. Our goal is to find such PV that will perform the best as a contextual teacher in our teacher-student training. And so, in Section 4.5.2, we pick several PV models and for each we find the best performing contextual loss given our structural loss. We also show results of models trained with just the contextual loss to illustrate how important is structural teacher to the performance of the student model. Finally, in Section 4.5.3, we experiment with the weighting of the two losses for the most promising student models.

4.5.1 Optimizing Paragraph Vector's training

We do not implement Paragraph Vector, instead we use implementation from Gensim library[4]. In this section, we explore some of the hyperparameters that govern the training of PV. From the best performing models, we choose few, which we use as contextual teachers when experimenting with the contextual loss in the subsequent section. We focus on four hyperparameters that we consider the most important, and adopt the recommendation of either the library or related literature. Both the adopted and the grid-searched hyperparameters are enumerated in Table 4.5, where the mentioned hyperparameters have the following meaning:

- `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 used PorterStemmer implemented by `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

[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 train 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. The rest of the hyperparameters we adopted either the default values set by the authors of the Gensim library[4] or recommended by the mentioned literature.

- `dbow_words` – whether to train word embeddings using Word2Vec [Mikolov et al., 2013] Skip-gram architecture simultaneously with DBOW document embeddings
- `epochs` – number of iterations over the corpus

As recommended by the authors of PV [Le and Mikolov, 2014], we train experiment with both architectures. For each architecture we try different settings of `vector_size`, `min_count` and `text_pre_process`, which all control the regularization of the model. Settings such as higher dimensional embedding, small minimum count and no text pre-processing, regularize the model the least. They give the model the most information on its inputs, while also providing it with large embedding through which the model can express precisely. On the other hand, using lower dimensional embedding, large minimum count and heavy text pre-processing, 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 to a small vector. We did not see any value in trying dimensions of embeddings higher than 1024 as in later experiments we need to distill the contextual embedding to a 768-dimensional embedding of our student model. Intuitively, the larger the contextual embedding is going to be, the smaller the percentage of information the student model is going to be able to digest. Also there is no value in considering `min_count` to be lower than 2, since we would only add words that are unique to a single document. Embeddings of such words would be poorly trained and would not add meaningful information to the document's embedding. Last hyperparameter that deserves mention is `dbow_words`. DBOW on its own does not train word embeddings, which are randomly generated. Setting `dbow_words` to true causes DBOW to also train word embeddings using Word2Vec Skip-gram model [Mikolov et al., 2013] in each epoch. Lau and Baldwin [2016] showed that random word embeddings significantly hurt the model, and so in our opinion the overall longer training, which training word embeddings inevitably causes, is worth the performance gain.

We train all variants on `val_500k` corpus. We follow the recommendations of Paragraph Vector's authors and evaluate also the concatenation of both architec-

tures. We pick the three best models from each architecture and we do a cartesian product of the two sets giving us 9 compound models. In total there are 45 models, whose performance on validation tasks we report in Figure 4.5. Among the single models, large embedding dimensions and low minimum count were favored. Additionally, on average, stemming or lowercasing lead to higher score than not pre-processing the words at all. DBOWs vary more in performance, occupying the best and the worst positions, whereas DMs are a bit more consistent. We achieve a slight improvement by concatenating a DM and a DBOW model, but considering the resulting model has effectively double the vector size, 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.

In our preliminary experiments we saw that the dimension of the contextual teacher's embedding plays a significant role in the teacher-student training. And 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`).

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 to give the student model more freedom in how it encodes information in 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 a linear projection, such that the correlation of two sets of vectors under their individual projections is maximal. We define CCA in Equation 1. CCA gives the student the freedom to change its embeddings as long as their linear projections correlate with the linear projections of the contextual teacher's embeddings.

Definition 1 (Canonical Correlation Analysis for more dimensions)**.** *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*

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

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 adjust the amount of freedom we give to the student model. Since CCA does not allow such adjustment, we focus on *Deep CCA* (*DCCA*) [Andrew et al., 2013]. DCCA projects the input vectors with two neural networks, and then

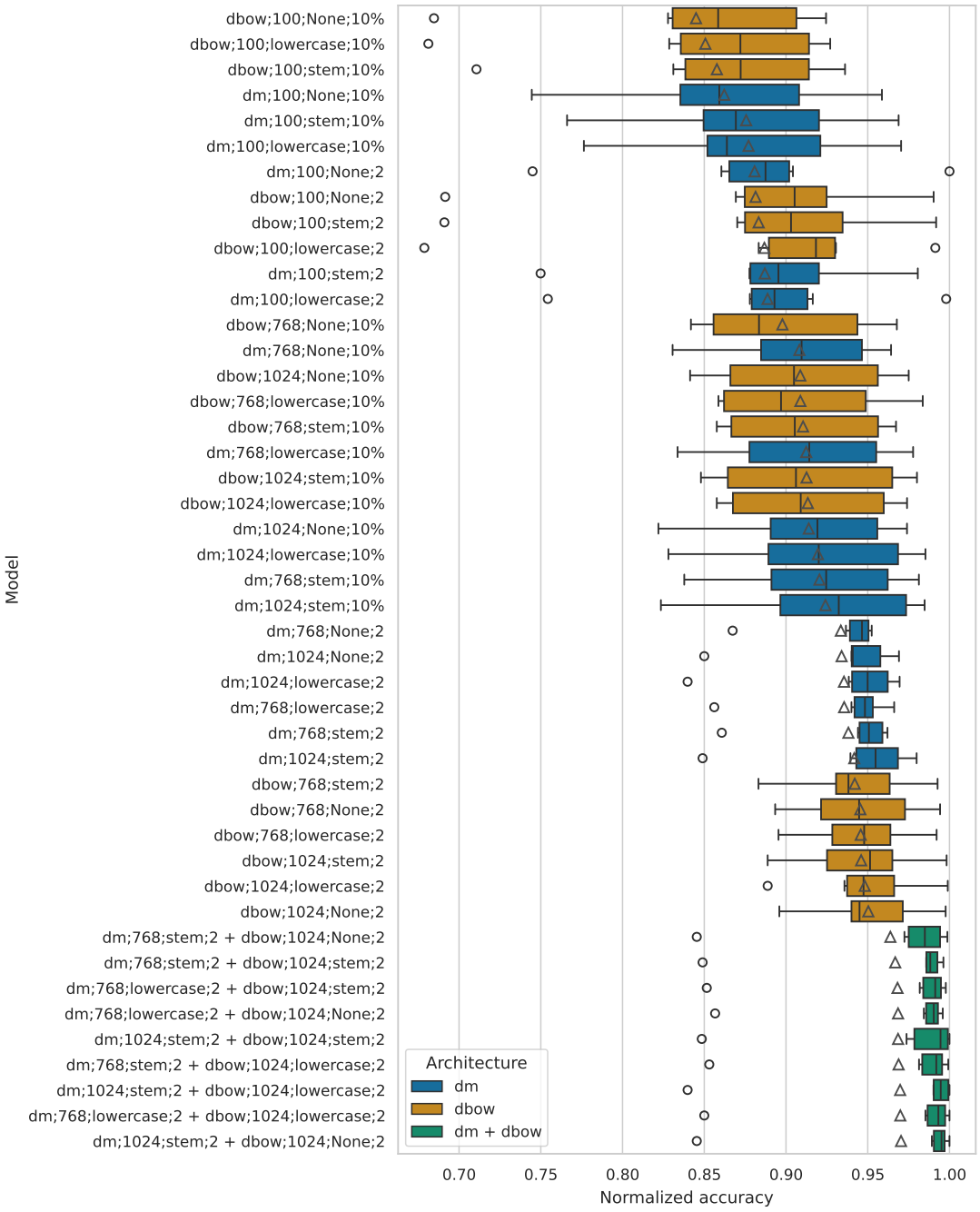


Figure 4.5: Performance of all grid-searched Paragraph Vector variants on validation tasks. A model is identified by architecture, embedding dimension, text pre-processing and minimum count, in this order. Models composed of two submodels are identified as a concatenation of such identifiers separated by +. The models are sorted according to the mean normalized accuracy marked with a triangle.

feeds the projections to the vanilla CCA. The two networks are simultaneously trained together 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 pressure the contextual loss inflicts on the student model. The larger the projecting neural network is, the more it is going to be able to transform the embeddings and the less the student model is going to need 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 amount of GPU memory and therefore are applicable only to small models. For this reason we avoid DCCA and use SoftCCA [Chang et al., 2018], which reformulates CCA, such that it is usable even in case of mini-batch optimization with small batches. To explain how SoftCCA is related to CCA we reformulate the solution of 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 with uncorrelated features. SoftCCA mimics these two effects with two separate losses:

- L2 loss which minimizes of the difference between projections:

$$\mathcal{L}_{\text{L2}}(X_1, X_2) = ||X_1 - X_2||_F^2 = \text{MSE}(X_1, X_2) \tag{4.9}$$

- *Soft Decorrelation Loss* (*SDL*) which forces a projection to have decorrelated features:

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

  where

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

Where

- $X, X_1, X_2 \in \mathbb{R}^{b \times d}$ are mini-batches of d-dimensional vectors
- Σ_X is a covariance matrix of a mini-batch of vectors X
- $\mathbf{0}_d$ is a $d \times d$ zero matrix
- β is a hyperparameter
- $\Phi_X^t, X^t, \hat{\beta}^t$ is $\Phi_X, X, \hat{\beta}$ at iteration t

The L2 loss forces the projected vectors to be equal, while the Soft Decorrelation Loss forces them to have uncorrelated features. Notice that to have a better estimate of the projected vector's covariance matrix, SDL keeps a running mean. 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.

$$\mathcal{L}_{\text{SoftCCA}}(X_1, X_2) = \mathcal{L}_{\text{L2}}(X_1, X_2) + \delta(\mathcal{L}_{\text{SDL}}(X_1) + \mathcal{L}_{\text{SDL}}(X_2)) \tag{4.15}$$

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 and then apply SoftCCA loss, which provides training signal not only for the embedding model, but also for the neural networks projecting the embeddings. For brevity we call the neural networks that project the student's and the contextual teacher's embedding *student* and *contextual projections*, respectively. We illustrate the used contextual loss graphically in Figure ref (TODO: image).

In our preliminary experiments, we found that the value of δ makes little difference and 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 a correct value of β showed 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 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 two projections and SoftCCA, while finding the ideal contextual projection without any structural loss. In the subsequent sections we study how different structural losses influence the ideal projection. First we experiment with cosine and then with max-marginals MSE loss. In all cases we test the projections for each of the three contextual teachers we selected in previous section. At the end, we select the best contextual loss with the best contextual teacher for both cosine and max-marginals MSE structural loss, with which we experiment further later in this chapter.

	Contextual teacher's embedding dimension		
Projection	100	1024	2048
Student	`768(ReLU)x1024(ReLU)x768`	`768(ReLU)x1024`	`1024(ReLU)x2048`
	`768`	`1024`	`2048`
Contextual	`100(ReLU)x768`	`768x1024`	`1024x2048`
	`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 student models overall.

Contextual projection with contextual loss only

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

As we learned in our preliminary experiments over parameterization of the projections hurts the performance of the model. Even though large projections result in smaller SoftCCA loss, they tend to harm the CCA of the student's and contextual teacher's embeddings. Strong projections compensate for the student's flaws, which lessens the pressure on the student model as it does not need to adjust its embedding that much. As a consequence, the student model learns very little compared to the projections. Similarly, strong contextual projections can take away pressure from the student projection and vice-versa. In this regard, it is important to keep the contextual projection small, to put more pressure on the student's side where the gradients can propagate to the embedding model itself.

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 information contained in a embedding throught the projection. With even larger final projection dimension, some of the final vector's features would have to rely on the same embedding's features due to the Pigeonhole principle. Consequently some of the final features would be inevitably correlated. This would create unnecessary conflict with the SDL loss, which forces the projections to output uncorrelated features.

We build the projections as a sequence of blocks, where each block is composed of a fully connected layer and a optional Rectified Linear Unit ($ReLU$). In preliminary experiments, we also tried adding Dropout, Batch or Layer Normalization layers at different places in a block, but they have either negligible or negative effect 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 it is used. The whole projection is then identified 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 a dash. We present all the projections' variations we tested in Table 4.6.

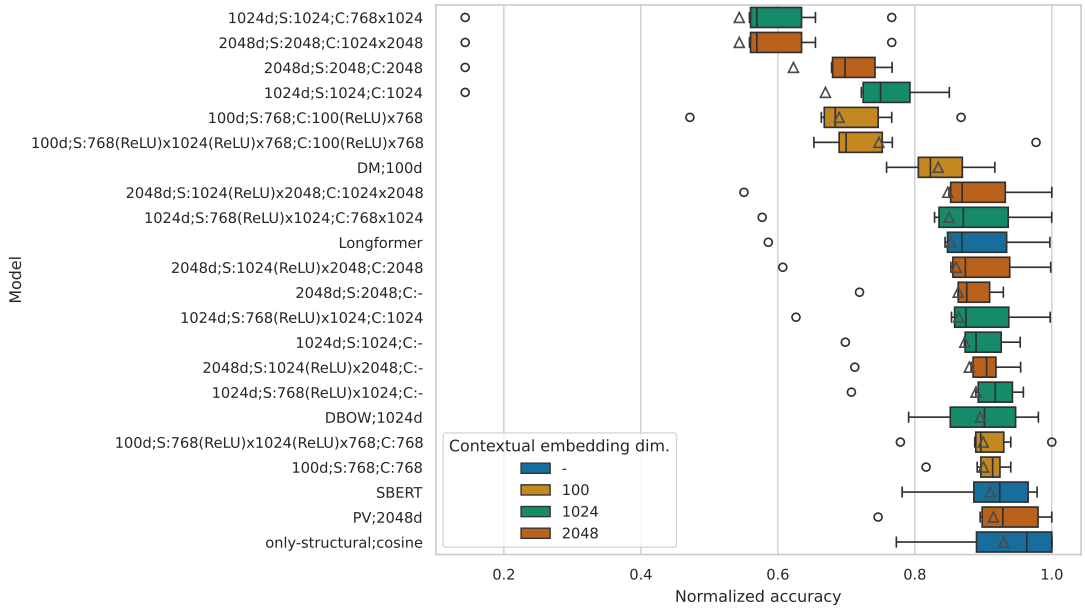


Figure 4.6: Performances of student models trained with only the contextual loss. We compare the students to all the relevant teachers, Longformer and the best student model trained with only the structural loss. We identify students trained with just the contextual loss with the contextual embedding dimension, student and contextual projection, in this order.

We train the student models on the first 15k documents of `val_500k`. We compare the models' performance to all the relevant teachers, Longformer and the best structural model in Figure 4.6. The results correspond to those we witnessed in our preliminary experiments and confirm the discussion from previous paragraphs. The better half of the student models differed from the rest by having a minimal contextual projection. Moreover, for a given contextual teacher and projection, the student model with larger student projection outperforms the student with a smaller one in all cases except for one.

Half of the tested projections improve the score of Longformer, suggesting that we are able to distill useful information from the contextual teacher to the student. However, as our contextual loss does not enforce exact similarity of the student's and the contextual teacher's embedding, most of the students do not outperform their respective contextual teachers. Interestingly, students trained with contextual embeddings with 100 dimensions surpass students trained with better performing 1024-dimensional contextual embeddings. The same can be said for 1024 and 2048 dimensional embeddings, even when comparing projections that increase with the size of the contextual teacher's embeddings, such as `1024d;S:1024;C:-` and `2048d;S:2048;C:-` or `1024d;S:1024;C:1024` and `2048d;S:2048;C:2048`. Consequently, it seems that it is easier to distill information from an embedding with less features than from a larger one. As a consequence, 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. And so in our experiments 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.

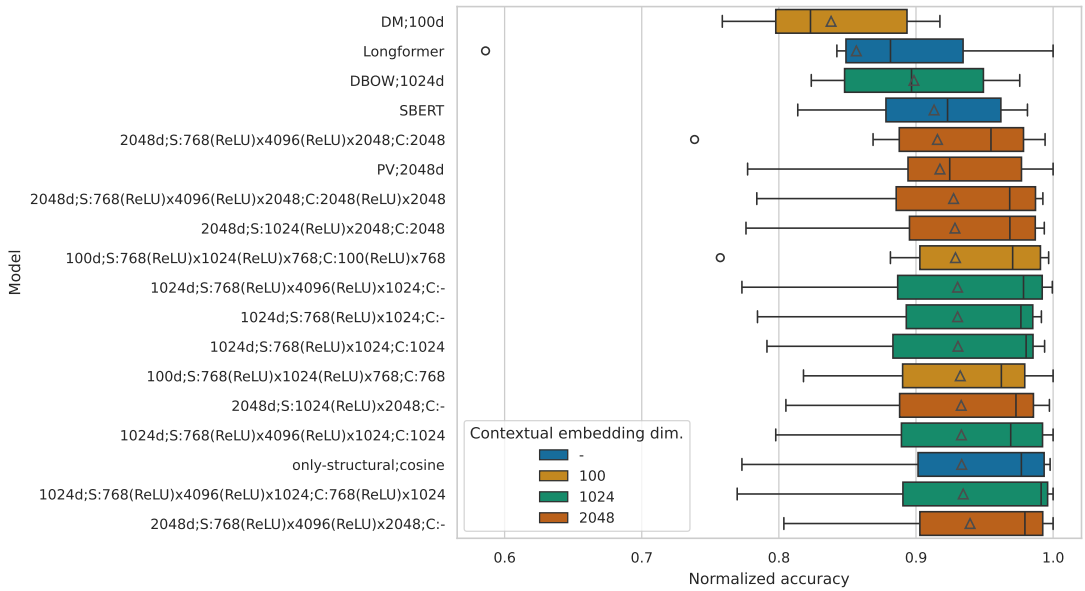


Figure 4.7: Performance of student models trained with contextual and cosine structural loss on validation tasks. We compare the student models to all relevant teachers, Longformer and the best student model trained with just the structural loss.

Contextual projection with cosine structural loss

We now search for the best performing projections while simultaneously using cosine structural loss. Compared to the previous section, the training is a bit more complex as the student is now forced to distill two qualities, each from a different teacher. So, even though the student and contextual projections still behave in the same manner, they may cause different outcomes.

We list all the tested projections in Table 4.7. We added 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 4.7. Compared to the student models trained with just the contextual loss, the same projection variants differ less with the cosine structural loss. By using the cosine structural loss, all of the student models get a performance boost that seems to lessen the negative effect of bad performing projections. 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. It seems that as we increase the pressure on the student by adding a structural loss, we need to simultaneously give the model more freedom on the side of the contextual loss. With the larger projections we were able to surpass the best student model trained with just the structural loss. This shows that, with well performing projections, the student model benefits from the fact that both losses are being used during training. Even if the performance gain is not huge, we can say the contextual and structural embeddings complement each other.

	Contextual teacher's embedding dimension
Projection	100
Student	768(ReLU)x1024(ReLU)x768
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	1024	2048
	-	-
Student	768(ReLU)x4096(ReLU)x1024	768(ReLU)x4096(ReLU)x2048
Contextual	768(ReLU)x1024	2048(ReLU)x2048
	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 a projections group. This gives 12 student models in total.

Contextual projection with max-marginals MSE structural loss

We also search for the optimal projections when training with the max-marginals MSE loss. Altough, as we show in Section 4.4.1, max-marginals MSE structural loss goes against the goal of our student-teacher training, it performed the best out of all composite and simple structural losses. We continue experimenting with it here, to see how does the student react if we also include the contextual loss.

We present all the tested projection variants in Table 4.8. We included successful projections from previous section and add stronger contextual projections as they perform surprisingly well with max-marginals MSE structural loss. Same as in the previous two sections, we train all student models on the first 15k documents from `val_500k` and compare their performance to all relevant teachers, Longformer and the student model trained with just the given structural loss. We present the model's performances in Figure 4.8. As is the case with the cosine strucutral loss, max-marginals MSE loss also boosts the students' performances. Again, as a consequence, we see that the student's performances are not as dependent on the projections as is the case of students trained without any structural loss. Contrary to what we witness in previous experiments, stronger contextual projections perform overall very well. Even if the best two models have weaker contextual projections, the majority of the above average student models, projects the contextual teacher's embedding with two layers. Whereas the majority of the below average student models, has one or no contextual layers.

	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 1024 -	1024x2048 2048 -
Student	768(ReLU)x4096(ReLU)x1024	768(ReLU)x4096(ReLU)x2048
Contextual	1024(ReLU)x1024 768(ReLU)x1024	2048(ReLU)x2048 -

(b) 1024 and 2048 dimensional contextual teachers

Table 4.8: All tested variants of projections with contextual loss and max-marginals 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 projections group. This gives 14 student models in total.

This suggests that max-marginals MSE exerts so much pressure on the student's embedding, that the contextual projection needs to be strong enough to conform to it. However, based on the two best performing projections, we see that an even larger student projection might compensate for a small contextual projection. Nonetheless, max-marginals MSE loss is so strict, it is more difficult to design the contextual loss so that the student benefits from it beeing used along the structural loss.

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. First, we assign static weights to each loss. Second, we combine the static weights with dynamic masking of inputs based on their length. As the structural teacher has limited context length, its embedding only reflects the information in the first 385 tokens. We use dynamic masking to train the student only on those inputs, which the structural teacher encoded whole. Therefore, in theory, the structural loss should be more reliable. To summarize, there are two parameters which we grid search: `max_structural_len` and λ. `max_structural_len` determines which inputs' structural loss we mask-out. λ is the static weight that is used for unmasked inputs to balance the importance of structural and contextual loss. For clarity, we include a Python-like pseudocode of the weighting algorithm:

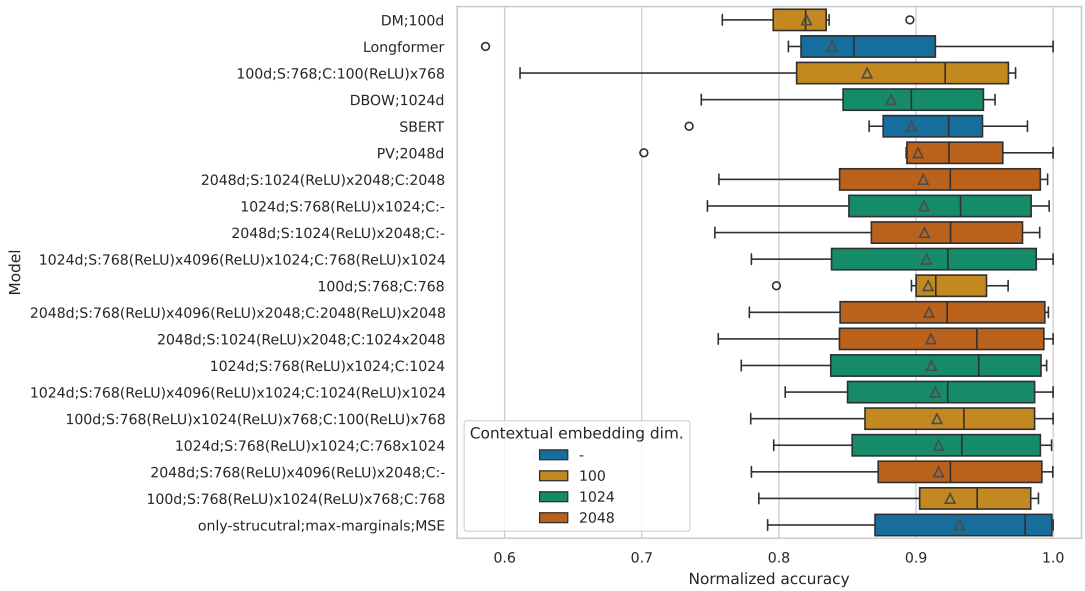


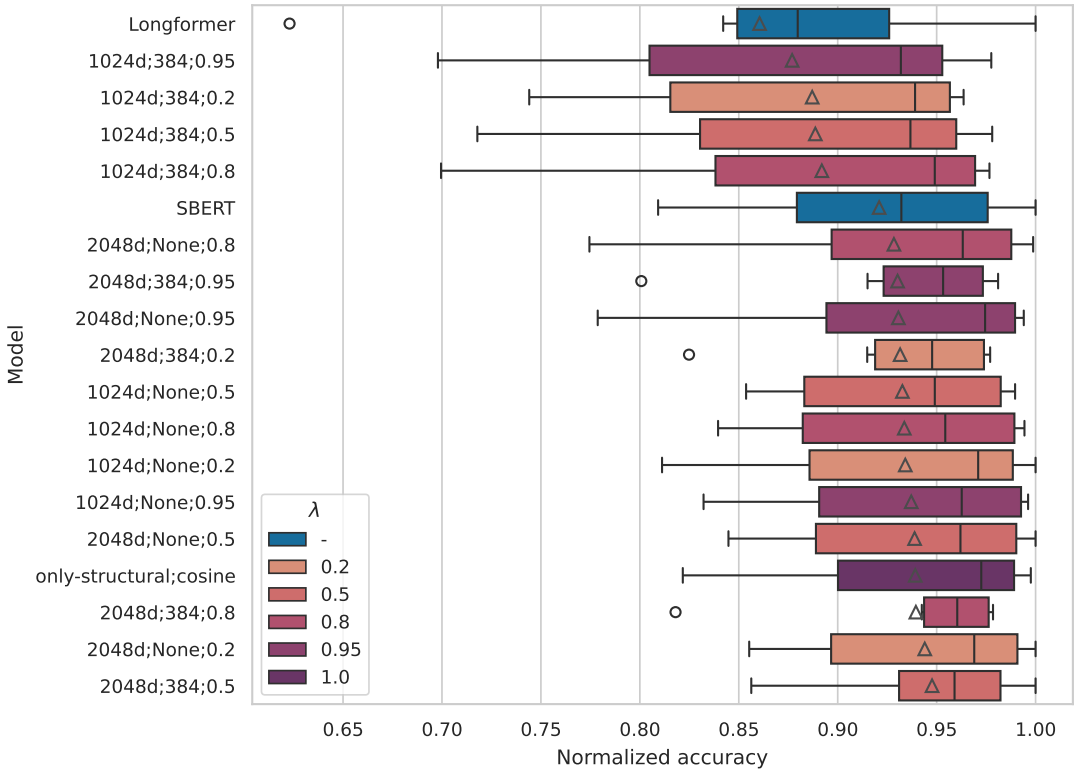
Figure 4.8: Performance of student models trained with contextual and max-marginals MSE structural loss on validation tasks. We compare the student models to all relevant teachers, Longformer and the student model trained with just the max-marginals MSE structural loss.

`max_structural_len`	λ
384	0.95
None	0.8
	0.5
	0.2

Table 4.9: Tested weighting hyperparameter values. We test all value combinations.

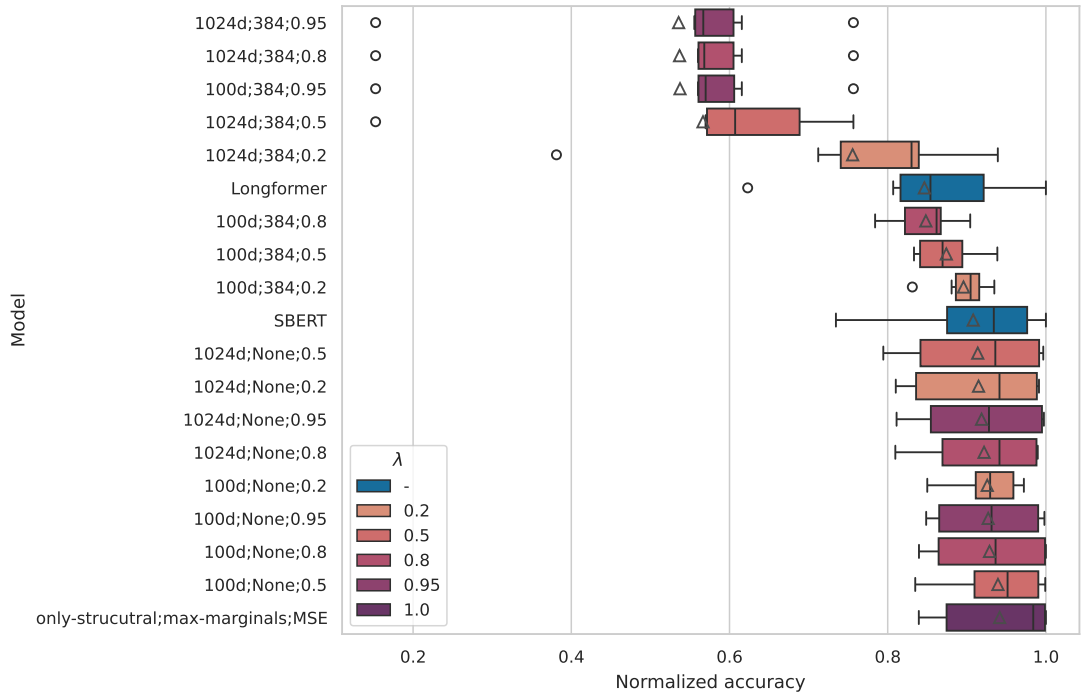
```
length_mask = torch.ones(batch_size)
if max_structural_len is not None:
  length_mask = lengths <= max_structural_len
lams = torch.zeros(batch_size).fill_(λ)
lams *= length_mask

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

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

We consider two structural losses: cosine and max-marginals MSE. For each structural loss we take the two best-performing contextual losses from the previous section and try all combinations of hyperparameters' values we list in Table 4.9.



Weighting a contextual and the cosine structural loss

Weighting a contextual and the max-marginals MSE structural loss


1024d;384;0.95
1024d;384;0.8
100d;384;0.95
1024d;384;0.5
1024d;384;0.2
Longformer
100d;384;0.8
100d;384;0.5
100d;384;0.2
SBERT
1024d;None;0.5
1024d;None;0.2
1024d;None;0.95
1024d;None;0.8
100d;None;0.2
100d;None;0.95
100d;None;0.8
100d;None;0.5
only-strucutral;max-marginals;MSE
Model
λ
-
0.2
0.5
0.8
0.95
1.0
0.2
0.4
0.6
0.8
1.0
Normalized accuracy

5. Evaluation

In this chapter we evaluate our student model on a number of downstream tasks. We train a student model with the best performing hyperparameters' values from Chapter 4 on a large text corpora. We then evaluate this final model on a set of 8 document downstream tasks. We simulate a scenario where there are no data to train a dedicated model on or finetune a pre-trained embedding model. In our opinion this is where a generic pre-trained embedding model is most valued and we therefore focus on evaluating embedding models for this exact use case. We also use the same checkpoint for all types of tasks to highlight the usability of our model in different use cases.

5.1 Our model

TODO: think of a name, so we are not constantly labelling it "our"

5.1.1 Training data

We train our student model on a corpus of long documents. We train our student model on the same data that were used to train Longformer. By using the data our student model was already trained on, we eliminate the possibility that the training data were the key ingredient and the cause of the performance gain or loss of our student model. Instead, our training method is to be blamed or given credit. Following Longformer's approach, we use the English Wikipedia[1] and documents from Realnews dataset [Zellers et al., 2019] that have above 1200 Longformer's tokens. We mix both sources equally, by randomly sampling from both datasets. Because Wikipedia does not have a validation split, we sample the validation documents again from its train split, avoiding already chosen training documents. We label the resulting dataset as `train_1500k` to clearly identify it. As can be seen in the Table 5.1, `train_1500k` contains 1.5 million long documents out of which only about 30% can be processed by our structural teacher whole. Though `train_1500k`'s length distribution, which is depicted in Figure 5.1, is far from uniform.

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

Split	Train	Validation
Documents	1 500 000	30 000
Tokens	2.06e+09	4.15e+07
Avg. tokens per document	1374.18±1736.64	1382.12±1697.09
SBERT tokens over 384	70.54%	70.73%
SBERT tokens over 512	66.36%	66.84%

Table 5.1: Statistics of `train_1500k`. For each split we also show the percentage of documents with the number of SBERT tokens above given threshold.

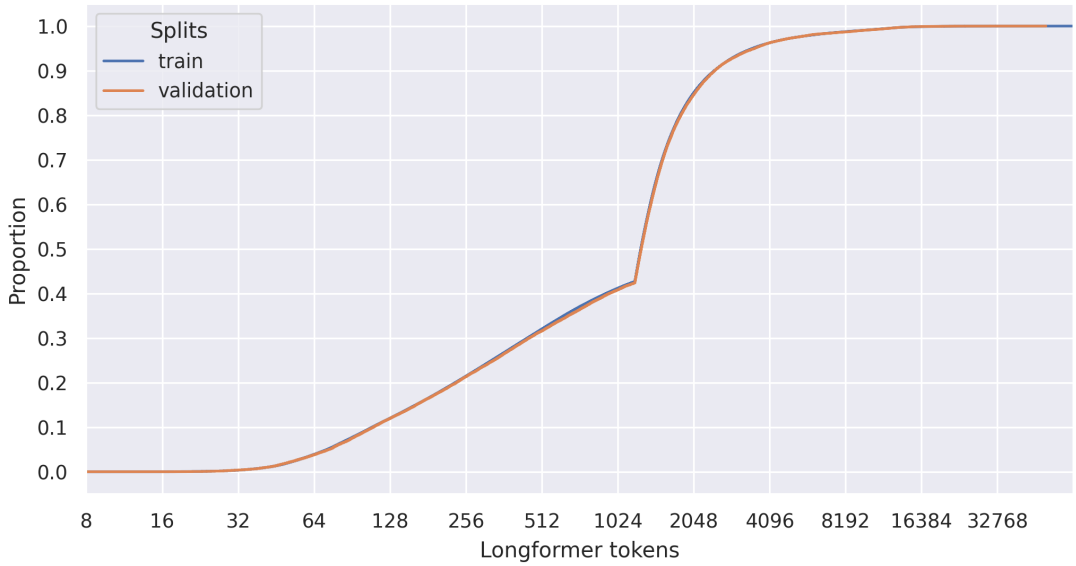


Figure 5.1: Estimated cumulative distribution function of number of Longformer tokens per document in `train_1500k`.

5.2 Classification tasks

Classification is a typical type of evaluation task for a textual embedding. In our evaluation set we cover several use cases including classification based on sentiment, on content, as well as citation and plagiarism detection. Our evaluation set also contains large span of document lengths from the very short texts having just over 100 tokens to extremely long with about 12 thousand tokens. An overview of our classification tasks is displayed in Table 5.2 and their splits' statistics in Table 5.3. We also plot the length distribution of all tasks in Figure 5.2 to show how well the tasks cover the range of document lengths.

We use a heavily regularized neural network as a classifier for each task. The classifier gets an embedding or a concatenation of two embeddings, in case of document pair classification, and is trained with a cross entropy loss to classify its input. We provide all the relevant hyperparameters in Table 5.4. For all task we use accuracy as the evaluation metric. For tasks with more classes we use micro averaging to give each input the same weight.

We do not finetune the embedding model for each task. While it could increase the performance, finetuning the embedding model for each task makes it more difficult to estimate the benefits of our training method.

5.2.1 Tasks' descriptions

IMDB movie reviews

The IMDB dataset [Maas et al., 2011] (denoted as IMDB) is a binary classification dataset that is quite frequently included in evaluation of long-context NLP models [Zaheer et al., 2020, Beltagy et al., 2020, Le and Mikolov, 2014]. The datasets consists of movie reviews, each with 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. For each movie

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%
OC	pairs of documents	2	50.00±0.07%	50.00±0.34%
AAN	pairs of documents	2	50.00±1.50%	50.00±0.77%
S2ORC	pairs of documents	2	50.00±0.09%	50.00±0.33%
PAN	pairs of documents	2	50.00±0.00%	50.00±0.00%

Table 5.2: Overview of our classification tasks. Each task classifies either documents or pairs of documents into several classes. We also show the mean and the standard deviation of percentage of class label within a given split.

Dataset	Split	Documents	Tokens over 384	Tokens 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%
OC	Train	240 000	12.31%	1.07%
	Test	30 000	12.24%	1.02%
AAN	Train	106 592	0.37%	0.06%
	Test	13 324	0.45%	0.08%
S2ORC	Train	152 000	33.24%	18.67%
	Test	19 000	32.96%	18.20%
PAN	Train	17 968	69.93%	59.45%
	Test	2 906	60.82%	47.37%

Table 5.3: Statistics of splits of our classification evaluation tasks. For each task and split we include the percentage of documents with SBERT tokens above a given threshold. This illustrates how many documents can our structural teacher process as a whole and how many documents need to be truncated.

Parameter	Value
Hidden features	50
Hidden dropout rate	0.5
Hidden activation	ReLU
Epochs	30
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 5.4: Parameters for training classification heads during evaluation.

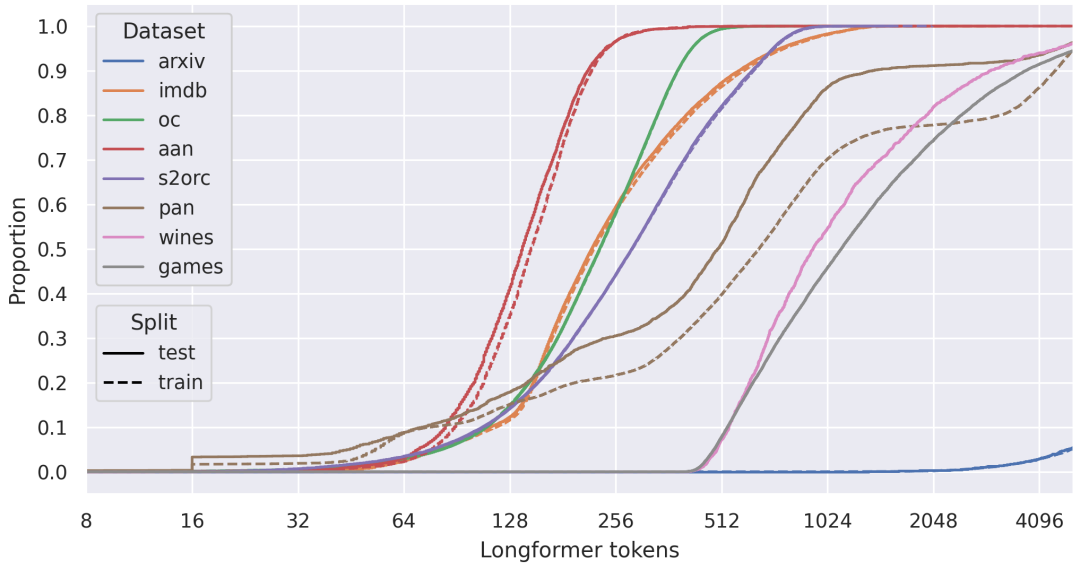


Figure 5.2: Estimated cumulative length distribution of the number of Longformer tokens in a document.

there can be up to 30 reviews and the test set contains disjoint set of movies. Along with the train and test splits the dataset contains also an unsupervised split without any rating or class labels. While the majority of the reviews are shorter; around 25% of documents are longer than the maximal context of our structural teacher and around 14% being longer than 512 tokens. Consequently, we use IMDB dataset to gain an estimate of the performance of our model on short to medium documents.

Arxiv papers

Arxiv papers [He et al., 2019] (denoted as ARXIV) is a collection of papers from ArXiv[2] – online archive of scholary papers. Each paper contains its full text spanning at least 1000 words. The longest papers are truncated to 10000 words. The papers are classified to 11 groups, that correspond to the scientific field of the paper. Since each paper can be associated with several scientific fields a small portion of the documents ($\approx 3.1\%$) appears more than once, but with different label each time. 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. The classes are not quite balanced with the most numerous class having around 12% of documents, while the least numerous class only having around 7%. ARXIV represents the extreme case of very long documents. As 97% of the dataset's documents are longer than the maximum context length of our student model.

ACL Anthology Network Corpus citations

The ACL Anthology Network Corpus citations dataset [Xuhui Zhou, 2020] (denoted as AAN) is a citation prediciton dataset. Each example in the dataset

[2] `arxiv.org`

contains a pair of paper's 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 Anothology Network Corpus [Radev et al., 2013], where each source paper creates positive pairs with all cited papers and negative pair with one other randomly sampled non-cited paper. The focus of this dataset is on very short documents; just around 4% of documents are longer than 256 tokens. AAN has the shortest documents of all datasets in our evaluation set.

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, which results in much higher count of unique papers in the dataset compared to AAN. OC contains relatively shorter sequences with only 12% of documents larger than the maximum context of our structural teacher.

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 prediciton dataset. The source of this dataset is the Semantic Scholar Open Research Corpus [Lo et al., 2019] where each paper is divided into sections that are connected via links to papers cited within the given sections. This structure is used to generate positive and negative pairs in the citation dataset S2ORC. A section is paired with an abstract of cited paper to create a positive pair and with an abstract of non-cited paper to create a negative pair.

The documents in S2ORC are comparatively longer than those of the previously mentioned tasks, except for ARXIV, with over 50% of document being longer than 256 tokens.

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 pair the source's plagiarised section with a part of the suspicious document containing the corresponding plagiarised section. Negative pairs are constructed from the positives one by replacing the source's segment by different part of the same document that is not annotated as being plagiarised. PAN contains longer documents; around 60% are longer than the maximum context size of our structural teacher and just under 50% of document are longer than 512 tokens. PAN therefore creates an intermediate step between documents of medium length in tasks such as IMDB or S2ORC and extremely long documents in ARXIV.

Dataset	Documents	Sources	Targets per source	Tokens over 384	Tokens 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 is composed of several source documents each with a set of similar target documents. Additionally we show the percentage of documents with SBERT tokens above a given threshold to highlight how many documents need to be truncated in order to be processed by our structural teacher.

5.2.2 Results

5.3 Similarity-based tasks

Besides from classification tasks we also evaluate our model on two tasks that are based on similarity metrics between two embeddings. Our evaluation set focuses on long Wikipedia articles from two different fields of interest. The statistics of similarity-based tasks are displayed in Table 5.5. The distribution of documents' lengths is plotted in Figure 5.2.

We use cosine as a measure of embeddings' closeness. Same as with the classification tasks, here we also do not finetune the embeddings to a given task. Consequently, we do not need training or validation splits and we can use the whole datasets for evaluation. Our main scoring metric is Mean Average Precision (*MAP*), but we also present Mean Reciprocal Rank (*MRR*). While MAP scores the whole predicted ordering, MRR is more interpretable and can be more important in scenarios where we care only about the first positive result.

5.3.1 Tasks' description

Wines and Video games Wikipedia articles

Both of our similarity-based tasks are datasets consisting of Wikipedia articles from two fields of interests: 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 10 similar articles. We find the two datasets very unique as they combine two aspects which are rarely seen together. First the similarities are based on human annotations and not on proxy measures such as common citations or outgoing links. Second the documents are relatively long with around 90% of documents being longer than 512 tokens.

5.3.2 Results

Conclusion

- what i have done — what are the model's results
- other findings

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.

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

A. Attachments